NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A HYBRID TIME SYNCHRONIZATION ALGORITHM BASED ON BROADCAST SEQUENCING FOR WIRELESS SENSOR NETWORKS

by

Sung C. Park

September 2014

Thesis Co-Advisors: Murali Tummala
John McEachen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2014	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE A HYBRID TIME SYNCHRONIZATION ALGORITHM BASED ON BROADCAST SEQUENCING FOR WIRELESS SENSOR NETWORKS	5. FUNDING NUMBERS
6. AUTHOR(S) Sung C. Park	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

In recent years, time synchronization has emerged as an essential research topic for wireless sensor networks. Numerous wireless network applications require a common reference time for collaborative data fusion and communications. Other time synchronization protocols have been proposed over the years, but none have eliminated the possibility of collisions associated with wireless packet transmissions while maintaining a precise level of synchronization for any given topology.

In this thesis, we present a new hybrid time synchronization scheme that provides a high degree of network-wide synchronization and eliminates the possibility of collisions when transmitting timestamp messages. We propose an algorithm that allows a network to determine a broadcast sequence by which nodes transmit and forward messages and then conducts a network-wide synchronization based on received timestamp information. The proposed hybrid time synchronization scheme utilizes two existing protocols, namely relative referenceless receiver/receiver synchronization and ratio-based synchronization protocol, that provide a high degree of precision. We implement our broadcast sequencing algorithm in simulations and demonstrate its effective performance for a series of network topologies. We also present results demonstrating an improvement in network-wide synchronization using our hybrid scheme over other time synchronization protocols.

14. SUBJECT TERMS wireless sensor networks, wireless ad hoc networks, time synchronization, broadcasting, clock synchronization	15. NUMBER OF PAGES 127
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A HYBRID TIME SYNCHRONIZATION ALGORITHM BASED ON BROADCAST SEQUENCING FOR WIRELESS SENSOR NETWORKS

Sung C. Park
Captain, United States Marine Corps
B.S., University of Illinois at Urbana-Champaign, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2014

Author: Sung C. Park

Approved by: Murali Tummala
Thesis Co-Advisor

John McEachen
Thesis Co-Advisor

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In recent years, time synchronization has emerged as an essential research topic for wireless sensor networks. Numerous wireless network applications require a common reference time for collaborative data fusion and communications. Other time synchronization protocols have been proposed over the years, but none have eliminated the possibility of collisions associated with wireless packet transmissions while maintaining a precise level of synchronization for any given topology.

In this thesis, we present a new hybrid time synchronization scheme that provides a high degree of network-wide synchronization and eliminates the possibility of collisions when transmitting timestamp messages. We propose an algorithm that allows a network to determine a broadcast sequence by which nodes transmit and forward messages and then conducts a network-wide synchronization based on received timestamp information. The proposed hybrid time synchronization scheme utilizes two existing protocols, namely relative referenceless receiver/receiver synchronization and ratio-based synchronization protocol, that provide a high degree of precision. We implement our broadcast sequencing algorithm in simulations and demonstrate its effective performance for a series of network topologies. We also present results demonstrating an improvement in network-wide synchronization using our hybrid scheme over other time synchronization protocols.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CRLB	Cramer-Rao lower bound
FTSP	Flooding time synchronization protocol
GPS	Global positioning system
MAC	Medium access control
NTP	Network time protocol
RBS	Reference broadcast synchronization
RSP	Ratio-based time synchronization protocol
RTC	Real-time clock
R4Syn	Relative referenceless receiver/receiver synchronization
TDMA	Time division multiple access
TPSN	Timing-sync protocol for sensor networks
WSN	Wireless sensor network
γ_{MR}	Master reference node
γ_{SN}	Starter node
γ_{Last}	Last node
γ_{LN}	Lone neighbor node
f_R	Relative clock skew matrix
Γ_C	List of children nodes
Γ_F	List of friend nodes
$\Gamma_{Neighbors}$	List of neighbor nodes
Γ_P	List of parent nodes
Γ_{RC}	List of remaining children
H_{min}	Minimum hop count
M	Number of broadcasts
N	Number of nodes
N_P	Number of parent nodes
O_R	Relative clock offset matrix

δ_{NB}	Network broadcast sequence
T	Reception timestamp matrix

EXECUTIVE SUMMARY

Over the past few decades, advances in technology have resulted in smaller and more efficient devices that enable communication among users. The need for these devices to be wireless has attracted significant interest in commercial and government applications. One of the popular wireless devices in demand is a sensor node that is capable of sensing, communicating, and processing data. Recent advances in the development of low-cost, low-power, multifunctional sensor nodes have led to the development of wireless sensor networks that cooperatively collect and disseminate sensor data. Wireless sensor networks consist of a large number of randomly deployed or self-organized sensor nodes that are appropriate for environments and operations that do not have a pre-defined network infrastructure, such as disaster relief or search and rescue operations. The attractive features of wireless sensor networks have extended research in a wide range of applications to include medical, environmental, and military operations.

One of the most critical components in the operation of wireless sensor networks is clock synchronization, as it provides a common time frame to different nodes. Most of the applications for wireless sensor networks require monitoring of events, thus requiring sensor nodes to maintain a common reference time for collaborative data fusion and communications. Signal-processing techniques that turn raw data into meaningful results require relative synchronization among sensor node clocks in order to detect a correct chronology of events. As the demand for wireless sensor networks in government and commercial industry increases, the broad application of time synchronization will become more prevalent as the requirement for data exchange increases.

The aim of this thesis is to develop, model, and simulate a new hybrid time synchronization scheme that provides a high degree of network-wide synchronization and eliminates the possibility of collisions when transmitting timestamp messages. Other time synchronization protocols offer accurate parameter estimators that synchronize nodes to an acceptable range but do not address the challenges associated with some network topologies. We present an algorithm that allows a network to determine a broadcast sequence by which nodes transmit and forward messages, eliminating collisions, and then

conducts a network-wide synchronization based on received timestamp information. Sensor nodes synchronize via one of two previously proposed time synchronization protocols, namely relative referenceless receiver/receiver synchronization and ratio-based synchronization protocol, that boast the highest level of precision in current research; thus, we present a hybrid scheme.

Significant improvements in network time synchronization were achieved in this thesis by our new hybrid scheme. Simulation models using MATLAB were developed to validate estimators proposed in previous research and to evaluate the performance of our broadcast sequencing algorithm as well as our new hybrid time synchronization scheme. Results in this thesis demonstrate that our algorithm produces a final broadcast sequence that includes a high percentage of nodes and that the sequence can be utilized to synchronize a network of any topology more precisely than other known time synchronization protocols.

We made three contributions in this thesis. We introduced a novel concept of broadcast sequencing that determines a sequence order and eliminates the possibility of collisions. We introduced new methods of determining a reference node to synchronize to and performing multi-hop synchronization using an analysis of time stamp information. Finally, we introduced a new hybrid time synchronization scheme and validated its superior performance over other time synchronization protocols.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Murali Tummala, for his exceptional mentorship and guidance throughout my academic time at the Naval Postgraduate School. His knowledge and ideas not only shaped the foundation of this work, but also brought much refinement throughout the process. I am sincerely grateful for his professionalism and dedication to our success, and will have the utmost respect for him for years to come.

I would like to thank my co-advisor, Dr. John McEachen, for his support of my research that made this thesis possible.

Above all, I would like to thank my beautiful, loving, and supportive wife for enduring the countless hours spent away for me to complete this thesis. She will never read this, but thank you for always preparing a delicious dinner for me to come home to.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Over the past few decades, advances in technology have resulted in smaller and more efficient devices that enable communication among users. The need for these devices to be wireless has attracted significant interest in commercial and government applications. One of the popular wireless devices in demand is a sensor node that is capable of sensing, communicating, and processing data. Recent advances in the development of low-cost, low-power, multifunctional sensor nodes have led to the development of wireless sensor networks (WSN) that cooperatively collect and disseminate sensor data [1]. WSNs consist of a large number of randomly deployed or self-organized sensor nodes that are appropriate for environments and operations that do not have a pre-defined network infrastructure, such as disaster relief or search and rescue operations [2]. The attractive features of WSNs have extended research in a wide range of applications to include medical, environmental, and military operations [1], [2].

One of the most critical components in the operation of WSNs, or any networked system, is clock synchronization, as this provides a common time frame to different nodes [3]. Most of the applications for WSNs involve monitoring of events, requiring sensor nodes to maintain a common reference time for collaborative data fusion and communications [4]. Signal-processing techniques that turn raw data into meaningful results require relative synchronization among sensor node clocks in order to detect a correct chronology of events [4]. Another example is the commonly used medium access control (MAC) protocol, time-division multiple access (TDMA), which requires precise time synchronization for its scheduling algorithm [4], [5].

As the demand for WSNs in government and commercial industry increases, the broad application of time synchronization will become more prevalent as the requirement for data exchange increases. We present a new time synchronization scheme that has the potential to increase the precision of a WSN's time synchronization, enhancing the network's ability to facilitate applications that require synchronization and improving the network's performance.

A. RELATED WORK

Network time synchronization research began with the global positioning system (GPS) and the network time protocol (NTP) [6], [7]. Both GPS and NTP work well for traditional networks with few power restraints but do not scale well to WSNs as wireless sensor nodes do not have a large power capacity. Typically, most deployments of wireless sensor nodes lack a real-time clock (RTC) or access to accurate time, thus, time synchronization poses a challenge for WSNs.

Ganeriwal et al. [4] presented timing-sync protocol for sensor networks (TPSN), a time synchronization protocol for WSNs based on a two-way message exchange between a sender and a receiver, similar to NTP. A limitation that TPSN suffers from is the nondeterministic delay from the sender and the inability to account for relative skew corrections.

Maroti et al. [8] presented the flooding time synchronization protocol (FTSP), a time synchronization protocol for WSNs based on one-way broadcasting from a sender to multiple receivers. FTSP also suffers from the nondeterministic delay from the sender. Sheu et al. [9] presented ratio-based time synchronization protocol (RSP), which is based on FTSP but eliminates some delay time, resulting in improved accuracy. We incorporate RSP into our proposed scheme due to this improvement.

Elson et al. [10] presented reference broadcast synchronization (RBS), a receiver-to-receiver-based time synchronization protocol for WSNs that uses the exchange of the arrival times of a reference node's broadcast to synchronize two or more receivers, eliminating the nondeterministic delay from the sender. A challenge that RBS suffers from is that the dedicated reference node serves as a single point of failure; therefore, it is not appropriate for self-organized WSNs.

Djenouri [11] presented relative referenceless receiver/receiver time synchronization (R4Syn), a method based on RBS that distributes the reference function and uses proposed estimators to synchronize a network of nodes. Djenouri's method offers a high degree of synchronization accuracy but does not discuss how the order of the reference function (i.e., a broadcast sequence) is determined for multi-hop topologies.

In this thesis, we incorporate R4Syn into our proposed scheme and investigate the challenges associated with WSNs, such as wireless packet collisions and changes in topology, both of which affect a network's ability to synchronize.

B. OBJECTIVE AND APPROACH

The objective of this thesis is to develop and demonstrate a new time synchronization scheme that addresses the challenges of any type of wireless ad hoc network topology and improves the overall precision of the network's time synchronization.

We propose a new time synchronization scheme based on a broadcast sequence that determines the order by which nodes transmit timing information in an effort to eliminate collisions. With this broadcast sequence, nodes transmit messages and synchronize as a network using one of two different time synchronization protocols. We conduct MATLAB simulations to evaluate the performance of the broadcast sequencing algorithm and the new hybrid time synchronization scheme.

C. THESIS ORGANIZATION

Sources of clock synchronization error and an overview of fundamental time synchronization approaches are presented in Chapter II. Our proposal of a hybrid time synchronization scheme using the broadcast sequencing algorithm and two previously published protocols is presented in Chapter III. Results of simulation experiments that validate and evaluate the hybrid time synchronization scheme are presented in Chapter IV. The findings of this thesis and recommendations for future work on time synchronization for WSNs are discussed in Chapter V. Additional flow charts for determining broadcast sequences are included in Appendix A. Source codes for calculations and simulations are included in Appendix B.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In Chapter I, we mentioned the need for and importance of small, low-cost wireless sensors for many applications. In order for a network of wireless sensor nodes to effectively conduct operations, their clocks need to be synchronized as precisely as possible. Time synchronization provides a common notion of time across a distributed system for fundamental operations, such as data fusion, power management, transmission scheduling and localization protocols [3].

In the following sections, we discuss the different sources of error in time synchronization, such as clock drift and delay times, in wireless packet transmissions. We provide a brief overview of the fundamental time synchronization approaches currently in literature and discuss some of the protocols related to each approach. We conclude the chapter with an in-depth discussion of one of the more recently proposed time synchronization protocols, relative referenceless receiver/receiver synchronization (R4Syn), which we adopt in our proposed hybrid scheme.

A. SOURCES OF ERROR IN TIME SYNCHRONIZATION

We begin by describing the clock of a sensor node X as $C_X(t)$ where t is real time. For a crystal-quartz oscillator commonly used in sensor nodes, the frequency can vary up to 40 parts per million (ppm) [3]. These differences in frequencies cause clocks to gradually drift from each other over time. Additionally, if clocks are not perfectly tuned to the same start time initially, there is a clock offset O, which is the phase difference of two clocks. The clock relationship between Node X to another Node Y is represented by

$$C_Y(t) = f_{X,Y} \times C_X(t) + \mathrm{O}_{X,Y}, \tag{1}$$

where $f_{X,Y}$ and $\mathrm{O}_{X,Y}$ are the relative clock skew (frequency difference) and clock offset (phase difference) between Node X and Node Y, respectively [3]. If the two nodes were

perfectly synchronized, then $f_{X,Y}$ would equal 1 and $O_{X,Y}$ would equal 0. The goal of clock synchronization is to estimate $f_{X,Y}$ and $O_{X,Y}$ as accurately as possible so that Node *X* can adjust its clock to Node *Y* [3].

In a perfect world where there is no delay in message delivery, nodes can send their current time to other nodes for synchronization with absolute precision. Unfortunately, we face various delays that affect message delivery in wireless sensor networks and make clock synchronization a difficult task. These delays in message delivery include the following [3], [8]:

- *Send time*: Nondeterministic time used to build the message at the application layer on the sender's side. This includes delays caused by the operating system when processing the request.
- *Access time*: Nondeterministic time for the sender to access the transmit channel after reaching the medium access control (MAC) layer. Depending on the MAC protocol, this is the least deterministic component.
- *Transmission time*: Nondeterministic time for the sender to transmit the message at the physical layer.
- *Propagation time*: Deterministic time that a message takes to go from the sender to the receiver. This only depends on the distance between two nodes.
- *Reception time*: Nondeterministic time for the receiver to receive the message at the physical layer. This is the same as transmission time.
- *Receive time*: Nondeterministic time for the receiver to process the message and send to the application layer. This is similar to the send time.

In addition to these various delay times in packet transmissions, clocks are also affected by environmental conditions such as temperature and atmospheric pressure. Aging hardware and voltage changes can affect clock parameters as well [3].

B. EXISTING TIME SYNCHRONIZATION PROTOCOLS

In this section, we discuss fundamental approaches to clock synchronization in WSNs using timing messages to account for the various delays discussed in the previous section. We explore three different approaches and discuss some of the protocols that exist in current literature.

1. Two-way Message Exchange

In this conventional approach, a sender and receiver synchronize by exchanging timing messages with each other [3]. We illustrate this approach using Figure 1 where parent Node *Y* wants to synchronize with child Node *X*. For one round of message exchange, Node *Y* sends a synchronization message to Node *X* at t_1. Node *X* records its local time that it received Node *Y*'s message at t_2 and sends a reply message to Node *Y* at t_3. Node *Y* receives the reply message that contains the timestamps t_2 and t_3 at its local time t_4. Using the four timestamps, Node *Y* can calculate its relative offset to Node *X*. After *k* rounds of message exchanges, Node *Y* obtains a series of timestamps that passes through filters and statistical analysis to obtain an accurate estimator. This classic approach is the foundation of the NTP, which has kept the internet's clock ticking with accurate synchronization; however, NTP was designed for traditional computer networks and is not suitable for WSNs [3], [7].

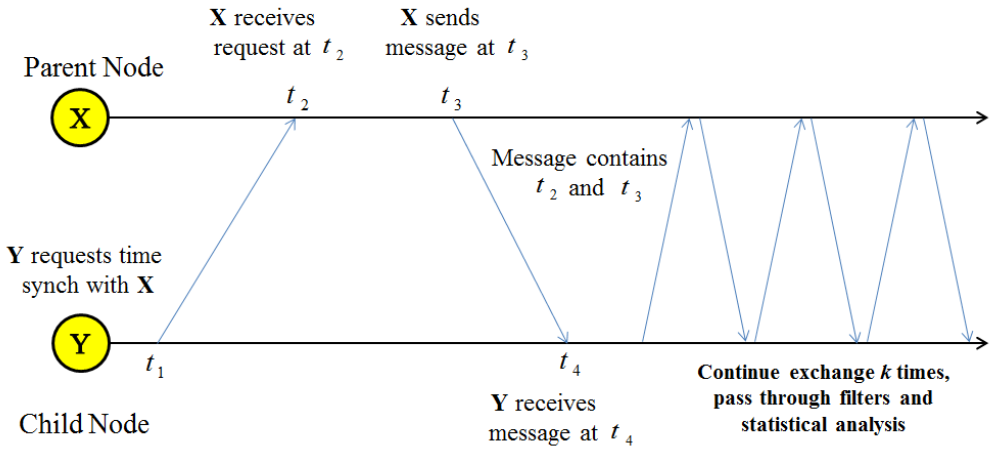

Figure 1. Two-way message exchange for time synchronization of two nodes (after [3]).

Another popular protocol that adopted the two-way message exchange approach for wireless sensor networks is timing-synchronization protocol for sensor networks (TPSN). TPSN works in two phases (a level discovery phase and a synchronization phase) to provide network-wide time synchronization based off of the two-way message

exchange approach [4]. In the level discovery phase, a hierarchical network structure is created, and then a pair wise synchronization is performed along the edges of the structure [4]. This establishes a global timescale throughout the network to which all nodes eventually synchronize their clocks during the synchronization phase [4]. Hardware experiments using Berkeley motes resulted in an average accuracy of less than 20 μs for a pair of neighboring nodes [4]. Some of our concerns, however, include the lack of relative skew corrections and the lack of broadcasting capability. Larger networks could potentially lead to higher costs.

2. One-way Dissemination

In this approach, a parent or reference node simply broadcasts its timing information to its children nodes. The receiving nodes record and collect the arrival times of the broadcast message as shown in Figure 2 and then compute relative parameters using least-squares estimation [3]. The flooding time synchronization protocol (FTSP) adopts this approach, where a single, dynamically elected parent node maintains the network time while other nodes synchronize their clocks to the parent node [8]. Hardware experiments on Berkeley motes resulted in an average precision of 1.5 μs in a single-hop scenario and 0.5 μs per hop in a multi-hop scenario [8].

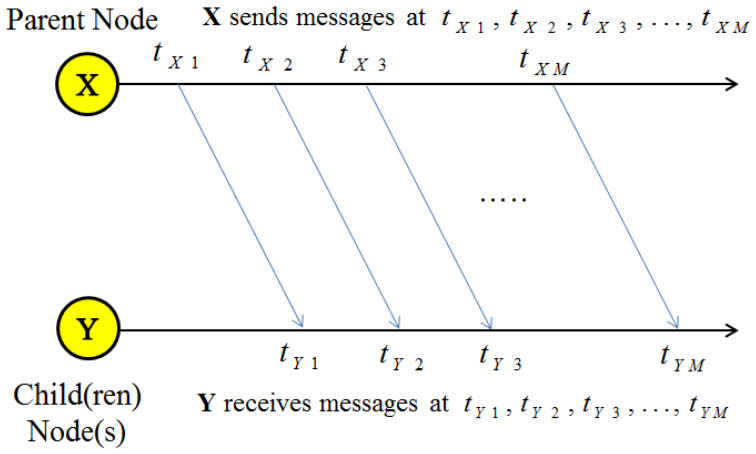

Figure 2. One-way dissemination (i.e., broadcasting) for time synchronization of multiple children nodes by one parent node (after [3]).

A more recently proposed broadcasting protocol, ratio-based time synchronization protocol (RSP), is similar to FTSP, but uses only two synchronization messages to synchronize receivers to the sender as shown in Figure 3 [9]. In RSP, the authors propose using a ratio of the timestamps to calculate the relative skew and offset, respectively, of Node *Y* to Node *X*, given by [9]

$$f_{Y,X} = \frac{t_3 - t_1}{t_4 - t_2} \tag{2}$$

and

$$O_{Y,X} = \frac{t_1 \times t_4 - t_2 \times t_3}{t_4 - t_2} \ . \tag{3}$$

The RSP method eliminates some delay time compared to FTSP, resulting in slightly improved accuracy during hardware experimentation [9]. Because of its improved accuracy and simple implementation, we incorporate the concept of RSP into our proposed hybrid time synchronization scheme in Chapter III.

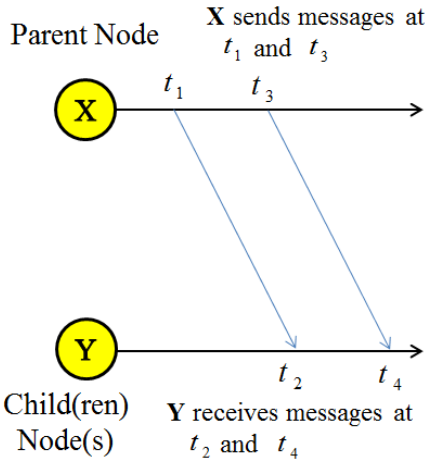

Figure 3. RSP message broadcasting (after [9]). A parent node synchronizes children nodes using two timing messages.

3. Receiver-receiver Synchronization

In this approach, a dedicated parent or reference node sends beacons while the receiving children nodes record the arrival times and exchange their arrival timestamps

with one another to calculate their relative skew and offset to the parent node [3]. The exchange of arrival timestamps is illustrated in Figure 4.

This approach, also called reference broadcast synchronization (RBS), uses the broadcast nature to reduce the time-critical path (i.e., the sender's nondeterministic delay) that results in high uncertainty and low accuracy [10]. Hardware experiments using Berkeley motes resulted in a clock precision of $1.85 \pm 1.28 \mu s$ for a pair of receivers and degraded as the number of hops increased in a multi-hop environment [10]. When RBS was compared to TPSN under the same scenario in [4], TPSN performed almost twice as well as RBS in regards to average synchronization error.

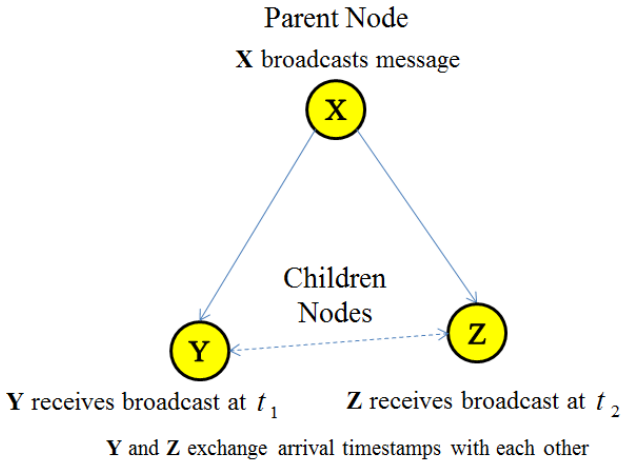

Figure 4. RBS synchronization (after [3]).

One of the major drawbacks of RBS is the dedicated reference node [11]. It serves as a single point of failure if it were to be eliminated, which is not appropriate for self-organized WSNs [11]. Additionally, RBS requires separate transmissions for exchanging timestamps, which increases the communications overhead [11]. More recently, however, a new protocol to address these concerns is proposed in [11], which we discuss in the next section.

C. RELATIVE REFERENCELESS RECEIVER/RECEIVER TIME SYNCHRONIZATION

We now formally introduce relative referenceless receiver/receiver time synchronization (R4Syn), a receiver-receiver-based protocol that attempts to address some of the drawbacks of RBS and serves as the primary method for the hybrid time synchronization scheme proposed in this thesis.

1. Methodology Overview

This protocol assumes that the network is static (mobility is not considered) and that nodes are located in the same vicinity of each other and are aware of their neighbor IDs. Unlike RBS, R4Syn distributes the reference function amongst all nodes, running in cycles where nodes sequentially broadcast beacons [11]. For a neighborhood of *N* nodes, each beacon piggybacks *N*–1 previously received timestamps. One cycle of beacon broadcasting is illustrated in Figure 5, where $B_{i,j}$ denotes the j^{th} beacon of node *i*, and $t_{i,j}^{k}$ denotes the reception timestamp at node *k* of the j^{th} beacon of node *i* [11]. For example, beacon $B_{4,j}$ would piggyback timestamps $t_{1,j}^{4}, t_{2,j}^{4}, t_{3,j}^{4}$ [11]. These timestamps are then used as samples to estimate relative skew and offset. More specifically, synchronization between Nodes 1 and 2 (i.e., Node 2's estimation of synchronization parameters with respect to Node 1) occur by gathering timestamp sample pairs $(t_{3,\mathrm{j}}^{1}, t_{3,j}^{2})$ and $(t_{4,\mathrm{j}}^{1}, t_{4,j}^{2})$ [11].

After a sufficient number of beacons, each node obtains the same number of timestamps that can be arranged into a matrix of reception timestamps,

$$T = \begin{bmatrix} * & t_{2,j}^{1} & t_{3,j}^{1} & t_{4,j}^{1} \\ t_{1,j}^{2} & * & t_{3,j}^{2} & t_{4,j}^{2} \\ t_{1,j}^{3} & t_{2,j}^{3} & * & t_{4,j}^{3} \\ t_{1,j}^{4} & t_{2,j}^{4} & t_{3,j}^{4} & * \end{bmatrix} \tag{4}$$

for one cycle of broadcasting, where the asterisk " * " denotes the broadcasting node. By observation, the location of a timestamp in the matrix *T* is the *k*-th row and the *i*-th column.

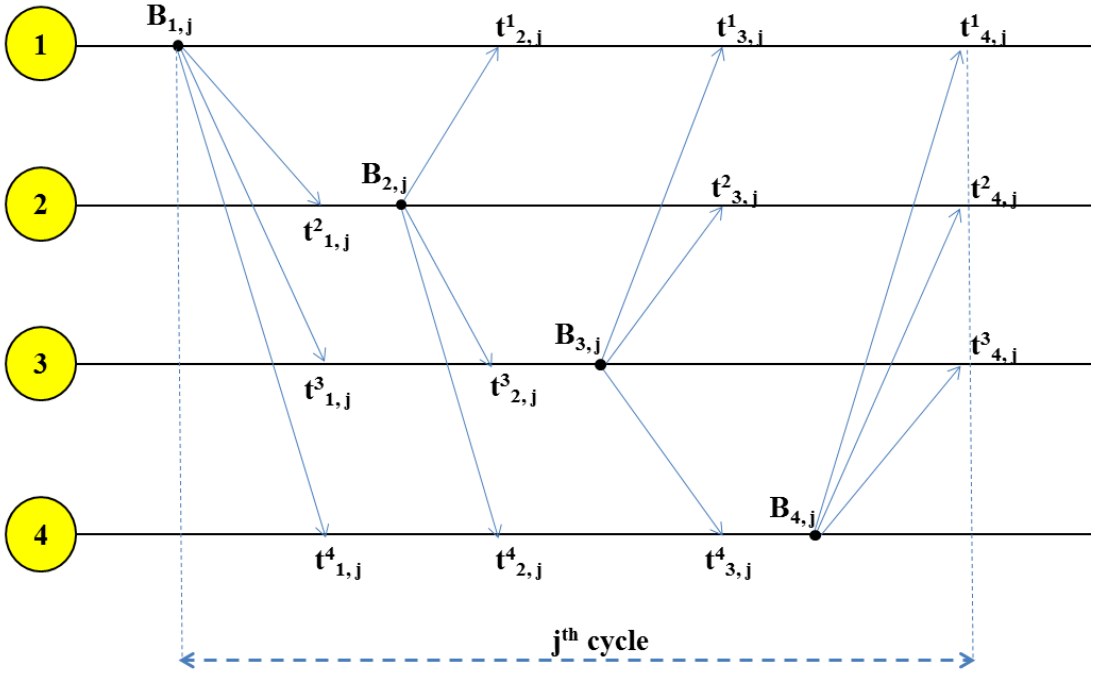

Figure 5. Example of R4Syn beacon broadcast during one cycle (after [11]).

If u_k and v_k, $k \in \{1,...,I\}$, is denoted as the k^{th} beacon reception timestamp of Nodes 1 and 2, respectively, then only beacons received by both nodes are used to construct timestamp samples, (u_k, v_k). Using maximum-likelihood (ML) estimation, we obtain the resultant estimates for relative skew and offset using I broadcast messages (i.e., timestamp values) as [11]

$$f_{ML} = \frac{\sum_{k=1}^{I} u_k \sum_{k=1}^{I} v_k - I \sum_{k=1}^{I} v_k u_k}{\left(\sum_{k=1}^{I} v_k\right)^2 - I \sum_{k=1}^{I} v_k{}^2} \tag{5}$$

and

$$\mathrm{O}_{ML} = \frac{1}{I}\left(\sum_{k=1}^{I} u_k - \frac{\sum_{k=1}^{I} u_k \sum_{k=1}^{I} v_k - I \sum_{k=1}^{I} v_k u_k}{\left(\sum_{k=1}^{I} v_k\right)^2 - I \sum_{k=1}^{K} v_k{}^2} \sum_{k=1}^{I} v_k \right). \tag{6}$$

For multi-hop environments, nodes perform multi-hop synchronization only on demand, where intermediate nodes forward local synchronization parameters to the

communicating nodes to allow them to calculate multi-hop parameters [11]. Consider that Node 1 needs to synchronize to a remote node q where the established route is $\{1,2,3,...,q\}$. Each intermediate Node h is assumed to already have estimated parameters relating it to the next Node $h + 1$ using the one-hop parameters in Equations (5) and (6) and forwards its estimates to Node 1. If we let $f_{j\to k}$ (respectively $\mathrm{O}_{j\to k}$) denote the relative skew (respectively offset) of Node j to Node k, then the multi-hop estimators for the example are given as [11]

$$f_{q\to 1} = \prod_{j=1}^{q-1} f_{j+1\to j} \tag{7}$$

and

$$\mathrm{O}_{q\to 1} = \sum_{j=2}^{q-1}\left[\left(\prod_{k=2}^{j} f_{k\to k-1}\right)\mathrm{O}_{j+1\to j}\right] + \mathrm{O}_{2\to 1}. \tag{8}$$

2. Advantages

Simulation results that compare the mean-squared error of the proposed estimators and the corresponding Cramer-Rao lower bounds (CRLB) are presented in [11]. Results show that the proposed estimators' mean-squared error decrease and approach the CRLB as the number of broadcast messages increases.

The mean-squared error for the multi-hop skew estimator was also simulated in a multi-hop linear network of 10 nodes in [11]. Simulation results demonstrated that an increase of the number of messages improves the precision even with a higher number of hop values. Furthermore, results show that the mean-squared error is kept at an acceptable range for a small number of messages as well as a large number of hops.

In addition to the performance of R4Syn's estimators, there are other advantages that the protocol offers over RBS. Because R4Syn distributes the reference function among all nodes, there is no single point of failure as is the case with RBS. R4Syn also allows timestamps to be piggybacked to beacons so that the need for separate transmissions for exchanging timestamps is eliminated. It exploits the broadcast property of RBS, which takes advantage of the precision achieved through receiver-receiver synchronization.

3. Drawbacks and Issues

Despite the attractive benefits that R4Syn offers, there are some potential flaws with the proposed protocol that are not addressed. We present these drawbacks using the topology in Figure 6, where nodes are depicted by yellow circles labeled numerically and the solid black lines indicate that a pair of nodes can communicate. Our main concern is related to the sequence by which the nodes broadcast their beacons for a self-organized network. In [11], node IDs are used to determine the order of the sequence and the beacons are broadcast sequentially, but how the order of the sequence is determined is not proposed. Consider a topology as in Figure 6 and assume that the order of the sequence is a simple ascending order of node IDs. If Node 1 starts the beacon process, then Node 3 never knows to broadcast because it never receives a beacon from Node 2. Even if Node 3 knew to broadcast upon receiving a message from Node 1, there is a possibility of a collision at Node 4 from Node 2's and Node 3's broadcasts. If we continue the sequence order to Node 10 (last node), there is no way for Node 1 to know when to broadcast again and begin the next cycle. Another issue that R4Syn fails to address is how the entire network gets synchronized (i.e., to which node all other nodes synchronize).

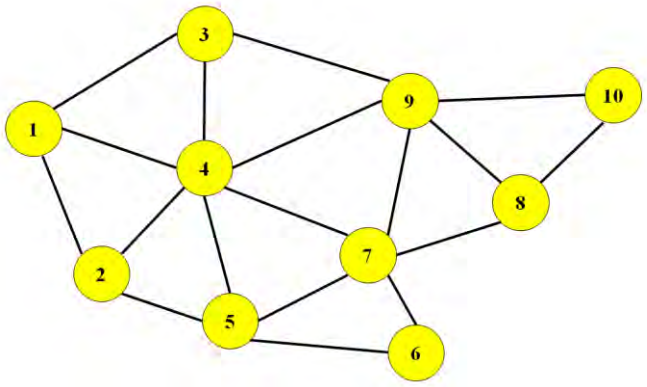

Figure 6. An example topology to illustrate drawbacks to R4Syn.

In this chapter, we first discussed the different sources of error in time synchronization, such as clock drift and delay times in wireless packet transmissions. We then provided a brief overview of the fundamental time synchronization approaches and

protocols currently in literature, one of which (RSP) we incorporate into our proposed hybrid protocol. Finally, we introduced R4Syn and discussed the advantages as well as the drawbacks that are not addressed in the literature. In the next chapter, we propose potential solutions to those drawbacks by presenting a new hybrid time synchronization scheme that significantly incorporates R4Syn's concepts.

THIS PAGE INTENTIONALLY LEFT BLANK

III. NETWORK TIME SYNCHRONIZATION IN MULTI-HOP WSN USING R4SYN-RSP HYBRID PROTOCOL

The crux of this thesis effort is the proposal of an innovative approach for an ad-hoc wireless sensor network to synchronize the nodes' clocks as precisely as possible. This new approach is rooted in the network's ability to determine a sequential order, or broadcast sequence, by which nodes transmit upon receiving a message from a neighbor node. This broadcast sequence allows the maximum number of nodes to transmit timestamp information sequentially in order to eliminate the possibility of collisions and to maximize the number of timestamp samples received by each node. As discussed in Chapter II, as a node acquires more timestamp samples, it is be able to generate more accurate parameter estimation for relative skew and offset. Using the broadcast sequence, we introduce a new method of collecting and analyzing timestamp information to determine how nodes are synchronized and, if required, synchronize any neighbor nodes. We formally introduce this new time synchronization scheme in this chapter.

A. PROPOSED TIME SYNCHRONIZATION SCHEME

We must begin the framework with a random deployment of unsynchronized sensor nodes that form an ad-hoc wireless sensor network. It is assumed that every node has the ability to communicate with at least one other node and discovery of one-hop neighbors is complete [2]. Our network-wide synchronization scheme consists of three major steps: determining a broadcast sequence, conducting a sequence correction check, and synchronizing via one of two existing protocols, namely R4Syn and RSP (see Chapter II).

We begin with a node we call the *starter node* γ_{SN} that is arbitrarily selected to initiate the broadcast sequence discovery and flood the network to begin the broadcast sequencing process. When the starter node γ_{SN} receives sequence information back from its one-hop neighbors, it determines the most node-inclusive broadcast sequence δ_{NB} for the network and broadcasts the sequence to proceed with the scheme. Upon receiving δ_{NB}, each subsequent node in the sequence performs a sequence correction check if required

and proceeds with its broadcast according to its location in δ_{NB}. The broadcast messages include all previous reception timestamps; thus. information forwarding is a key element to this proposed scheme. By simply examining the timestamp information, each node determines the *master reference node* γ_{MR}, the node to which the entire network attempts to synchronize. The network then conducts a hybrid time synchronization technique using R4Syn and RSP parameter estimation. This proposed scheme is summarized in Figure 7.

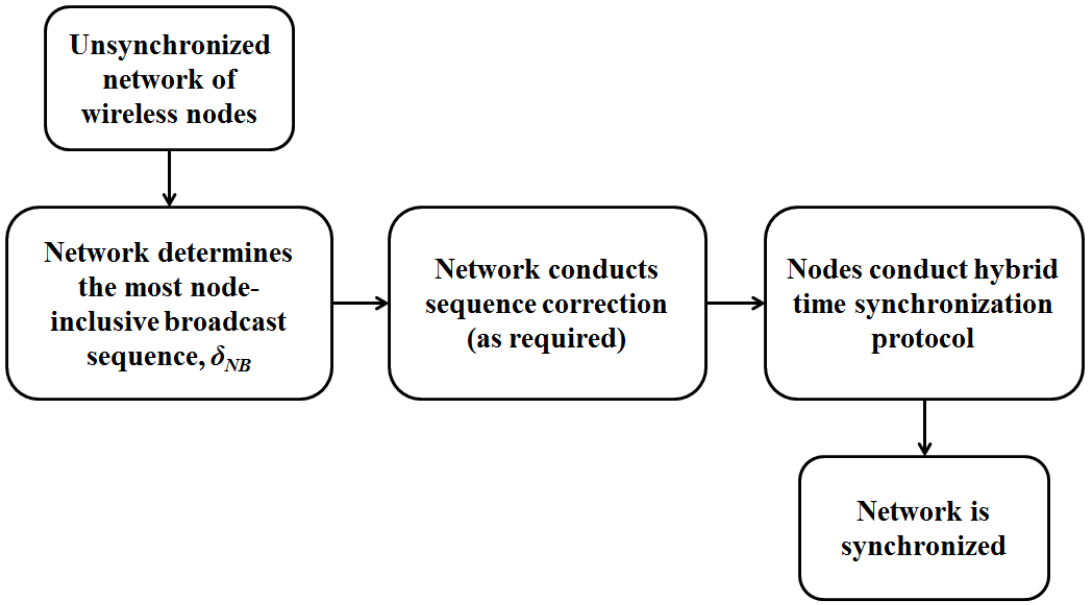

Figure 7. Proposed time synchronization scheme.

B. BROADCAST SEQUENCING

We now propose a solution to the sequencing and collision problems of R4Syn presented in Chapter II. In this section, we describe the intuitive concept behind discovering the most node-inclusive broadcast sequence δ_{NB}, discuss the flooding procedure that the starter node γ_{SN} initiates, describe how sequences are determined by each node, and provide flow diagrams to illustrate the logic behind sequence determination.

1. Broadcast Sequencing Concept

As we introduced in the previous section, an arbitrarily chosen starter node γ_{SN} initiates the protocol and floods the network after the nodes have been deployed. After the starter node γ_{SN} receives pertinent information back from its one-hop neighbors, it attempts to determine a broadcast sequence that includes the maximum number of nodes in the network. To illustrate this idea, we examine a topology in Figure 8. If Node 1 is selected to be the starter node γ_{SN}, then Node 1 determines a broadcast sequence δ_{NB} like {1, 2, 4, 3, 5}. Specifically, this means that Node 2 transmits after receiving Node 1's broadcast, Node 4 transmits after receiving Node 2's broadcast, Node 3 transmits after receiving Node 4's broadcast, and so on. If Node 4 was selected to be γ_{SN}, it still determines a δ_{NB} like {5, 3, 4, 1, 2} but not a sequence like {4, 3, 5, 2, 1}. In the latter case, not only does the sequence have an error since Nodes 2 and 5 are not neighbors, but also Node 2 never transmits because it never receives a broadcast from Node 5.

The main idea behind determining δ_{NB} is that whichever node is the starter node, δ_{NB} should include the maximum number of nodes in an order that nodes know when it is their turn to broadcast upon receiving a broadcast message from one of its neighbors. This eliminates any possibility of collisions because each node in δ_{NB} has its own turn to broadcast. When it is the broadcast turn of nodes in the beginning or end of δ_{NB}, those beginning/end nodes reverse the sequence to continue the broadcast cycle. More specifically, using the example of Node 1 as γ_{SN} with a δ_{NB} of {1, 2, 4, 3, 5}, Node 5 reverses the sequence to {5, 3, 4, 2, 1} and forwards that sequence during its turn so that Node 3 knows to broadcast next upon receiving the message from Node 5.

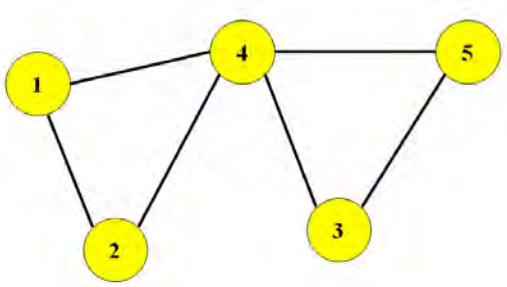

Figure 8. Example topology illustrating the intuitive concept behind determining a broadcast sequence that will eliminate collisions.

2. Flooding

The protocol begins with the starter node γ_{SN} flooding the network with an initialization message. In this flooding phase, nodes simply exchange hop count information with their one-hop neighbors as the initialization message propagates throughout the network. We denote the minimum number of hops from γ_{SN} to a given node X as $\mathrm{H}_{\min}(X)$. Clearly,

$$\mathrm{H}_{\min}(\gamma_{SN}) = 0. \tag{9}$$

After a Node X receives $\mathrm{H}_{\min}$ from all of its neighbors, it determines its own $\mathrm{H}_{\min}$ as

$$\mathrm{H}_{\min}(X) = \min(\mathrm{H}_{\min}(\Gamma_{Neighbors}(\mathrm{X}))) + 1, \tag{10}$$

where $\Gamma_{Neighbors}(X)$ is defined as Node X's list of neighbors. Unlike a sequence like δ_{NB} where the order matters, the order of nodes in a list does not matter. Given the minimum hop count information, nodes define one-hop neighbor relationships according to $\mathrm{H}_{\min}$. If $\mathrm{H}_{\min}(X) = k$ hops, then the list of the parents of X, $\Gamma_P(X)$, is defined as one-hop neighbor nodes with $\mathrm{H}_{\min} = k-1$ hops. The list of the children of X, $\Gamma_C(X)$, is defined as one-hop neighbor nodes with a $\mathrm{H}_{\min} = k+1$ hops. The list of the friends of X, $\Gamma_F(X)$, is defined as one-hop neighbor nodes with the same value of $\mathrm{H}_{\min}$ as Node X. The one-hop neighbor relationships are summarized in Table 1.

Table 1. Summary of one-hop neighbor relationships.

Minimum Number of Hops from γ_{SN} (H_{min})	**Relationship to Node *X***
$k-1$	*Parents* of *X*
k	*Friends* of *X*
$k+1$	*Children* of *X*

Using the topology example in Figure 9 where Node 1 is γ_{SN}, we see that Node 4 defines its lists of one-hop neighbors as $\Gamma_P(4)=[1]$, $\Gamma_F(4)=[2]$, $\Gamma_C(4)=[3,5]$.

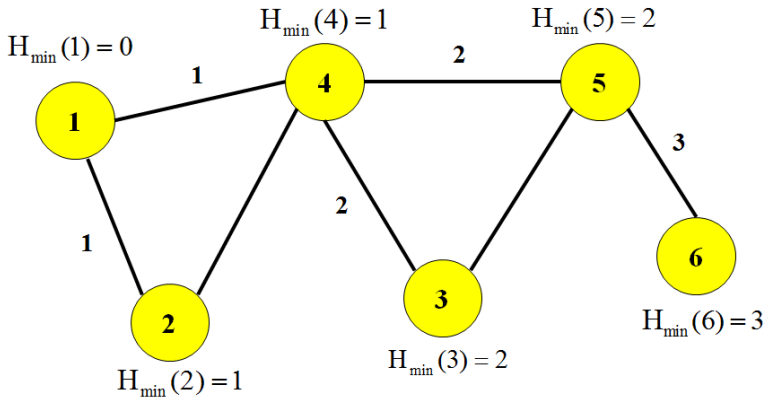

Figure 9. Topology illustrating minimum hop counts from the starter node, $\gamma_{SN}=1$. These hop counts are used to define a node's relationships to its neighbors.

3. Network's Broadcast Sequence

According to the parent-friend-child relationships defined in the previous section, we see that nodes without children determine their sequences first. For a childless node *L*, its sequence is simply defined as

$$\delta(L) \leftarrow \{L\}. \tag{11}$$

Next, the childless nodes broadcast their sequences to their parents, which also implies that all of the neighbors of the childless nodes receive the sequence information as well. Along with the sequences, those nodes send two additional items: their list of

friends $\Gamma_F(\mathrm{L})$ and their total number of parents $N_P(\mathrm{L})$. When a parent node receives sequence information from all of its children, it determines its own sequence and sends it to its parents, and so on. Parent nodes also send their list of children Γ_C in addition to their list of friends and the total number of parents. These items are used when a parent node has multiple children nodes with equal sequence/list lengths and needs to determine which child node's sequence is the most node-inclusive. Detailed flow charts that describe how Γ_C, Γ_F and N_P are used in sequence determination are included in Appendix A. This phase of the protocol is complete when the starter node γ_{SN} receives sequences from all of its children and determines δ_{NB} for the network.

Next, we describe how each node's sequence is determined in detail using the example topology of Figure 9, where Node 1 is the starter node γ_{SN}. From Equation (11), we can see that $\delta(2) \leftarrow \{2\}$, $\delta(3) \leftarrow \{3\}$, and $\delta(6) \leftarrow \{6\}$ since those nodes do not have any children nodes. Those three nodes broadcast their sequences, list of friends, and number of parents immediately after determining their own sequence. Per the relationship definitions from the previous section, we see that Node 1 is a parent to Nodes 2 and 4, Node 4 is a parent to Nodes 3 and 5, and Node 5 is a parent to Node 6.

Beginning with Node 5, we see that it has only one child node, Node 6. Node 6's sequence includes only its own ID, thus, Node 5 determines that

$$\delta(5) \leftarrow \{\delta(6), 5\}. \tag{12}$$

Since $\delta(6) = \{6\}$, Node 5's sequence becomes

$$\delta(5) = \{6, 5\}. \tag{13}$$

Node 5 then broadcasts $\delta(5)$, $\Gamma_F(5)$, $\Gamma_C(5)$, and $N_P(5)$ to its neighbors because it has received sequences from all of its children.

After Node 4 receives this broadcast from Node 5, it can determine its own sequence. Because Node 4 has two children nodes, Nodes 3 and 5, we begin by defining

$$\Gamma_{RC}(4) = \Gamma_C(4) \tag{14}$$

where *RC* is a short notation for *remaining children* so that $\Gamma_{RC}(4) = [3,5]$. Node 4 then finds Node 5 to have the longer sequence of the two children nodes, so it takes on Node 5's sequence as its own: $\delta(4) \leftarrow \{\delta(5)\}$, or $\delta(4) = \{6,5\}$. Node 4's list of remaining children $\Gamma_{RC}(4)$ is updated by removing Node 5 from the list, so $\Gamma_{RC}(4) = [3]$. We define the *last node* of a sequence as

$$\gamma_{Last} = \delta(\mathrm{X})_{i=m} \text{ , } i = 1, 2, \ldots, m \tag{15}$$

where *i* is the index of the sequence of any node X, and *m* is the number of nodes in the sequence. So, in the case of Node 4, $\gamma_{Last} = 5$. Next, Node 4 updates its sequence by examining γ_{Last} and $\Gamma_F(3)$. Because γ_{Last} is a member node of Node 3's list of friends $\Gamma_F(3)$, Node 4 updates its sequence as follows:

$$\delta(4) \leftarrow \{\delta(4), 3\}. \tag{16}$$

Because Node 3 was added to $\delta(4)$, Node 3 is removed from $\Gamma_{RC}(4)$ and $\Gamma_{RC}(4)$ becomes an empty list; that is, the number of its remaining children is zero. Node 4 completes its sequence by appending itself at the end, $\delta(4) \leftarrow \{\delta(4), 4\}$ and, subsequently, broadcasts $\delta(4)$, $\Gamma_F(4)$, $\Gamma_C(4)$, and $N_P(4)$ so that its parent node, Node 1 (also the starter node γ_{SN}), can determine δ_{NB} for the whole network. Because of this network's topology, Node 1 follows the same algorithmic steps as Node 4 to determine that $\delta_{NB} = \{6,5,3,4,2,1\}$, which includes every node in the network. This makes it an all-node-inclusive sequence because every node has a turn to broadcast, generating more timestamp samples to use the R4Syn estimators and synchronize with more precision. The final sequences for each node can be seen in Figure 10.

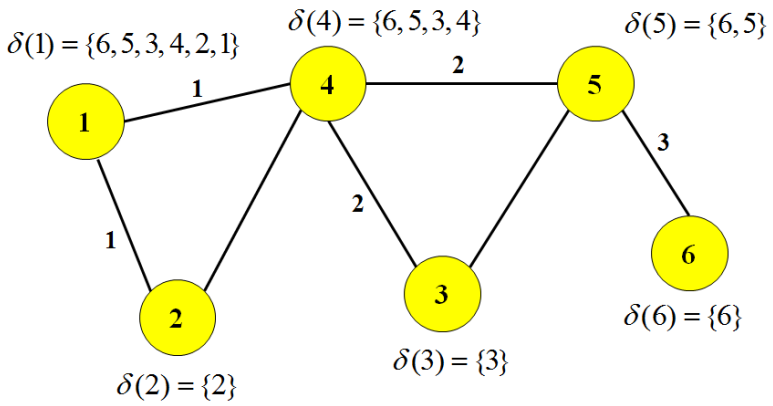

Figure 10. Example of each node's final sequences according to broadcast sequencing. The starter node $\gamma_{SN} = 1$ determined a sequence that included all nodes in the network.

Node 1 now commences the network time synchronization by first reversing δ_{NB} to $\{1,2,4,3,5,6\}$ and then broadcasting δ_{NB} as a message. The receiving nodes (Node 2 and 4) record the reception timestamps of Node 1's broadcast message. Node 2 knows to broadcast next because it is after Node 1 in δ_{NB} and Node 2 received the broadcast from Node 1. On the other hand, Node 4 knows to wait to broadcast until it receives a broadcast from Node 2, eliminating any possibility of a collision. As nodes receive subsequent broadcasts, they know when it is their respective turns to transmit depending on their location in δ_{NB}. Nodes use the reception timestamp information to conduct network time synchronization, which we discuss in the later sections with more detail.

4. Broadcast Sequencing Flow Charts

The sequence determination scheme described in the preceding sections can be represented by flow charts displayed in Figures 11 and 12 as well as those in Figures 39–45 in Appendix A. These flow charts provide the logical process behind the sequence determination algorithm at each node. In this section, we also introduce and explain additional variables that were not transparent in the previous sections.

When a node has only one child node, it checks to see where that child's ID is with respect to its own sequence per Figure 11. For example, if Node 2 has a child Node

3 with a sequence of $\delta(3)=\{6,5,1,3,2,4\}$, it sees that Node 3's ID is in the fourth index of this sequence. The sequence, δ_{LS} (short for *longer side*), is $\{6,5,1,3\}$ because it contains more nodes in that sequence than the right half of the sequence ($\{3,2,4\}$). To complete Node 2's sequence according to Figure 11, we get $\delta(2)=\{6,5,1,3,2\}$.

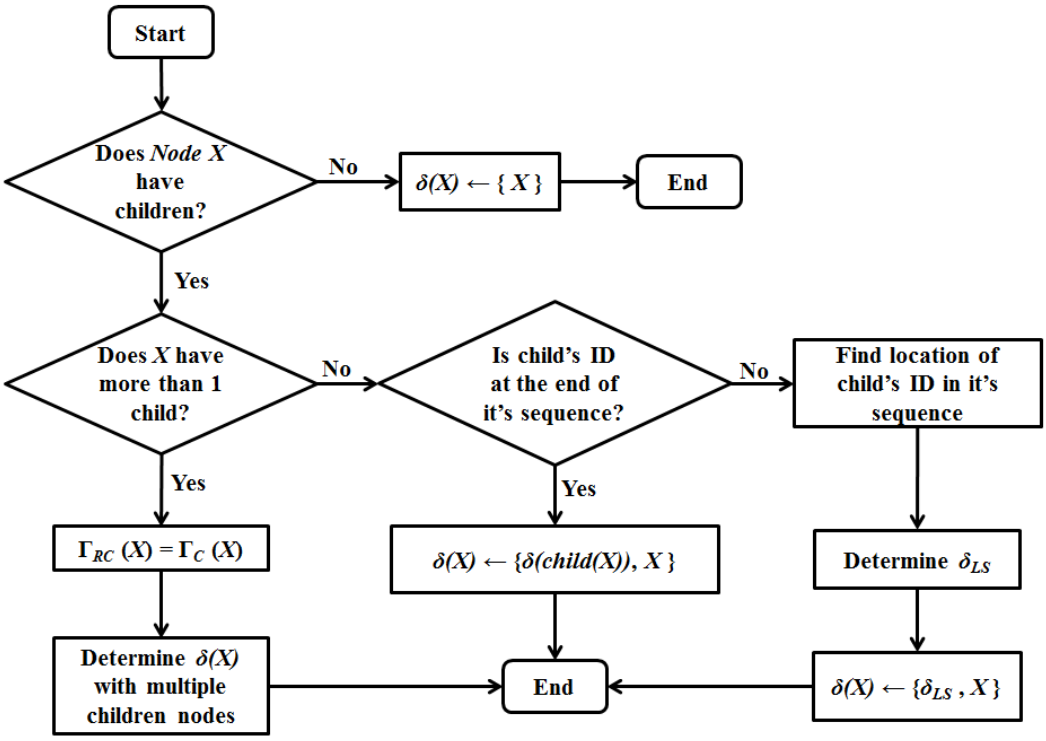

Figure 11. Flow chart illustrating sequence determination for nodes with less than two children nodes.

When a node has more than one child node, there are more algorithmic steps to determine its sequence. We now clarify the additional variables and functions found in the flow charts displayed in Figure 12 and Figures 39–45 in Appendix A.

We begin by using a generic node, say Node 5, with three children nodes: 1, 3, and 8, so $\Gamma_{RC}(5)=[1,3,8]$. We denote the child node with the longest sequence as Node *A*. In order to determine its sequence, Node 5 first chooses the child node *A*, say Node 1,

so $A = 1$. Say Node 1's sequence is $\{7,6,4,1,2\}$, so Node 5 updates its sequence with Node 1's sequence (i.e., $\delta(5) = \{7,6,4,1,2\}$).

Continuing with the flow chart in Figure 12, we denote δ_{Inc1} as Node *A*'s sequence up to and including its ID. For example, in Node 1's case, $\delta_{Inc1} = \{7,6,4,1\}$. The sequence δ_{Inc1} is potentially used in Node 5's sequence determination depending on the sequence of the child node *D* with the next longest sequence. The same concept applies to δ_{Inc2}, which we denote as Node *D*'s sequence up to and including its ID.

Next, Node 5 updates its list of remaining children by removing Node 1 from $\Gamma_{RC}(5)$ since Node 1 was added onto Node 5's sequence, so now $\Gamma_{RC}(5) = [3,8]$. Recall the last node γ_{Last} from the previous section except in this case, $\gamma_{Last} = 2$ since $\delta(5) = \{7,6,4,1,2\}$. Say Node 3 is a parent (or friend) of Node 2, so Node 3 is added onto Node 5's sequence (i.e., $\delta(5) = \{7,6,4,1,2,3\}$). This add parent/friend function is detailed further in Appendix A, where we assign the variable *PF* a logical 1 (i.e., *PF* = 1) if a node is added to the sequence during the function and a logical 0 otherwise (i.e., *PF* = 0). Node 5's list of remaining children is consequently updated to $\Gamma_{RC}(5) = [8]$ since Node 3 was added onto Node 5's sequence.

Continuing with the flow chart in Figure 12, Node 5 checks to see if its list of remaining children $\Gamma_{RC}(5)$ is empty, which it is not. So Node 5 chooses Node 8 as *D*, the child node with the next longest sequence (which, in this case, Node 8 is the only child node left and has the next longest sequence by default). Say Node 8's sequence is $\delta(8) = \{11,9,8,10\}$. Node 5 then sets the next sequence δ_{Next} as Node 8's sequence (i.e., $\delta_{Next} = \{11,9,8,10\}$), and $\delta_{Inc2} = \{11,9,8\}$ according to the same logic as δ_{Inc1}. Next, Node 5 updates its list of remaining children by removing Node 8, making $\Gamma_{RC}(5)$ empty. Per Figure 12, the value $PF_{Next} = 0$. Node 5 then updates its sequence $\delta(5)$ and δ_{Next} according to the *PF* and PF_{Next} values, which is outlined in Appendix A. For this example, $\delta(5) = \{7,6,4,1,2,3\}$ since *PF* = 1, and $\delta_{Next} = \delta_{BL} = \{11,9,8\}$ since $PF_{Next} = 0$.

Next, Node 5 checks for similarities between its sequence and δ_{Next}, which it does not have. So Node 5 updates its sequence $\delta(5) \leftarrow \{\delta(5), 5, \text{reverse}(\delta_{Next})\}$ per Figure 12, where $\text{reverse}(\delta_{Next})$ is simply reversing δ_{Next} (i.e., $\{8,9,11\}$). Node 5's sequence thus becomes $\delta(5) = \{7,6,4,1,2,3,5,8,9,11\}$. Since Node 5's list of remaining children is empty, it completes its sequence determination.

If there are similarities between $\delta(5)$ and δ_{Next}, then Node 5 adjusts its sequence per the flow charts detailed in Figures 43–45 located in Appendix A. The input 1 in Figure 12 is a recursive step from some of the functions in Appendix A.

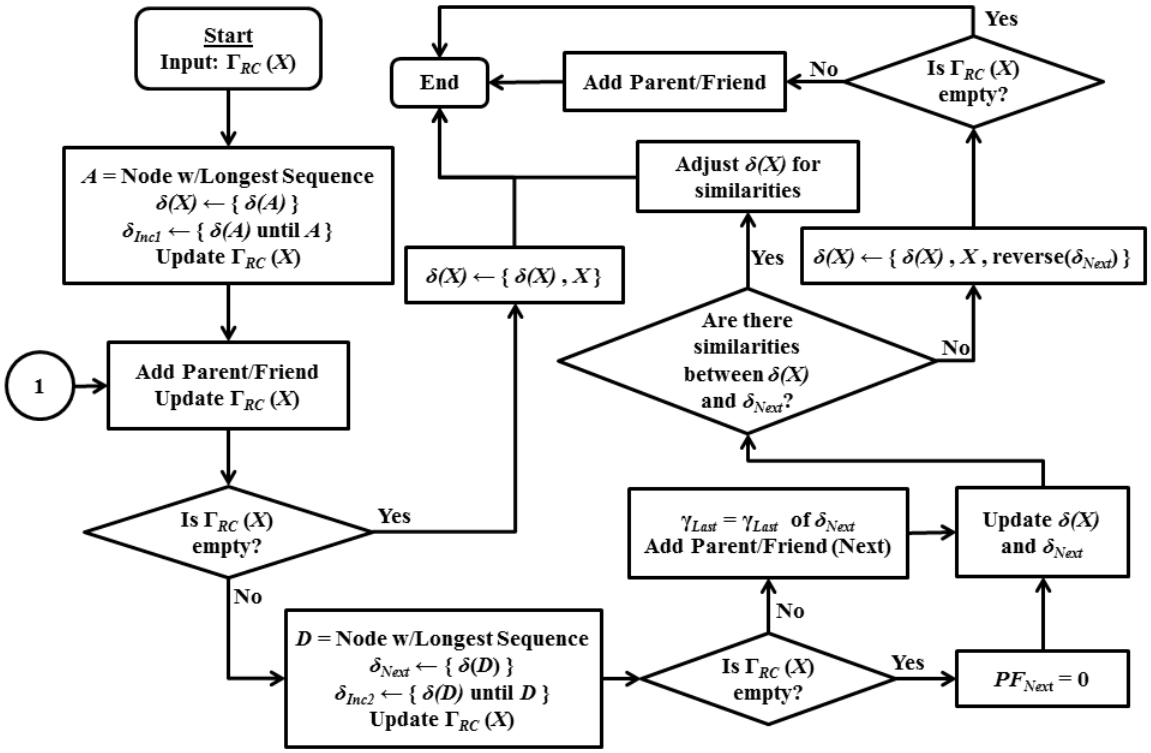

Figure 12. Flow chart illustrating sequence determination for a node with multiple children nodes.

C. SEQUENCE CORRECTION

In actuality, the sequence correction check can be conducted simultaneously with the network time synchronization as the initial broadcast sequence δ_{NB} is forwarded across the network because only reception timestamps of the broadcast messages are required for this. For our purposes, we discuss the sequence correction check as an independent step to prevent any confusion.

The purpose of the sequence correction is to add nodes into δ_{NB} that were potentially left out. As δ_{NB} is forwarded within the network, each node performs an additional sequence correction check before its broadcast. We illustrate the need for a sequence correction check with the topology in Figure 13. When Node 4 is the starter node γ_{SN}, it generates $\delta_{NB} = \{6,3,2,4,5\}$, which is not all inclusive because Node 1 is missing from the sequence. When Node 5 is the starter node γ_{SN}, it also generates a δ_{NB} that is not all inclusive. The proposed algorithm without a sequence correction measure produces a final δ_{NB} that is not all inclusive a third of the time for this topology. In order to alleviate this drawback, we propose an additional step for each node as the pre-sequence-corrected δ_{NB} is forwarded across the network.

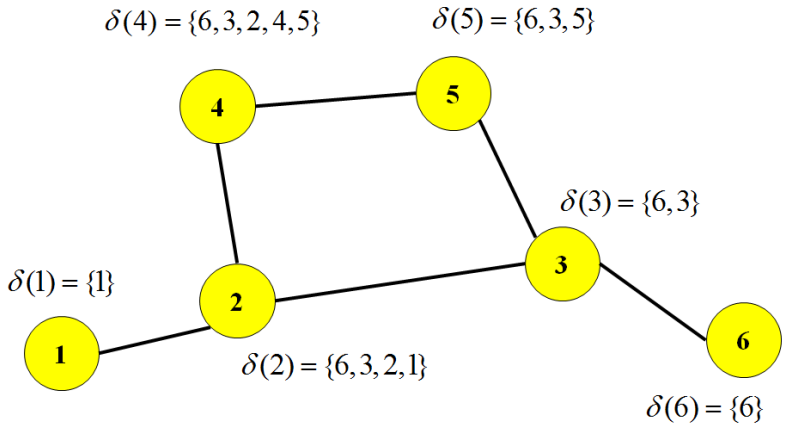

Figure 13. Topology illustrating the need for a sequence correction step. Node 4 does not determine a broadcast sequence that includes all nodes in the network.

In this example topology, Nodes 6 and 5 are at the beginning and end of δ_{NB}, respectively, and the neighbors of both nodes are included in δ_{NB}. If one of those nodes had a neighbor that was not included in δ_{NB}, they append that neighbor to the beginning or end of δ_{NB}; however, Node 1 is the node that is not in δ_{NB}, so we denote Node 1 as a *lone neighbor*, γ_{LN}. In this case, Node 2 adds a field to its broadcast message indicating that $\gamma_{LN} = 1$, which is forwarded throughout the network. Each node does a sequence correction check and either performs an action or ignores the γ_{LN} field. For this example, Node 5 knows that Node 3 is a neighbor node, so it adjusts δ_{NB} to $\{2,4,5,3,6\}$. Now that Node 2 is at the beginning of δ_{NB}, it can append Node 1 to the beginning of the sequence so that $\delta_{NB} = \{1,2,4,5,3,6\}$, which includes all nodes in the network.

A detailed flow chart of the sequence correction check for every node is shown in Figure 14. Using Figure 13 with Node 4 as the starter node, we examine Node 2's sequence correction per the flow chart in Figure 14. Say the order is from left to right in $\delta_{NB} = \{6,3,2,4,5\}$, so Node 3 is Node 2's previous node. Node 3 does not send a lone neighbor γ_{LN} because all of its neighbors are in δ_{NB}; thus, Node 3 does not have a γ_{LN} to send. Continuing on with the flow chart, we see that Node 2 checks to see if it is at the beginning or end of δ_{NB}, which it is not. Next, Node 2 finds a lone neighbor Node 1 ($\gamma_{LN} = 1$) and forwards that and δ_{NB} to the next node in the sequence, Node 4. Node 4 does the same and forwards $\gamma_{LN} = 1$ and δ_{NB} since Node 1 is not a neighbor and its ID is not at the beginning or end of δ_{NB}. When Node 5 receives $\gamma_{LN} = 1$ and δ_{NB} from Node 4, it sees that its ID is at the end of the sequence and adjusts δ_{NB} to $\{2,4,5,3,6\}$ because Node 3 is one of its neighbors. Forwarding of $\gamma_{LN} = 1$ and δ_{NB} continues until it reaches Node 2, which is now at the beginning of δ_{NB}. Node 2 knows that its neighbor, Node 1, is not in δ_{NB} and appends Node 1 to the beginning of δ_{NB} per the flow chart. Nodes continue to forward and use δ_{NB} in order to distinguish which node transmits when, eliminating any possibility of collisions as previously discussed in this chapter.

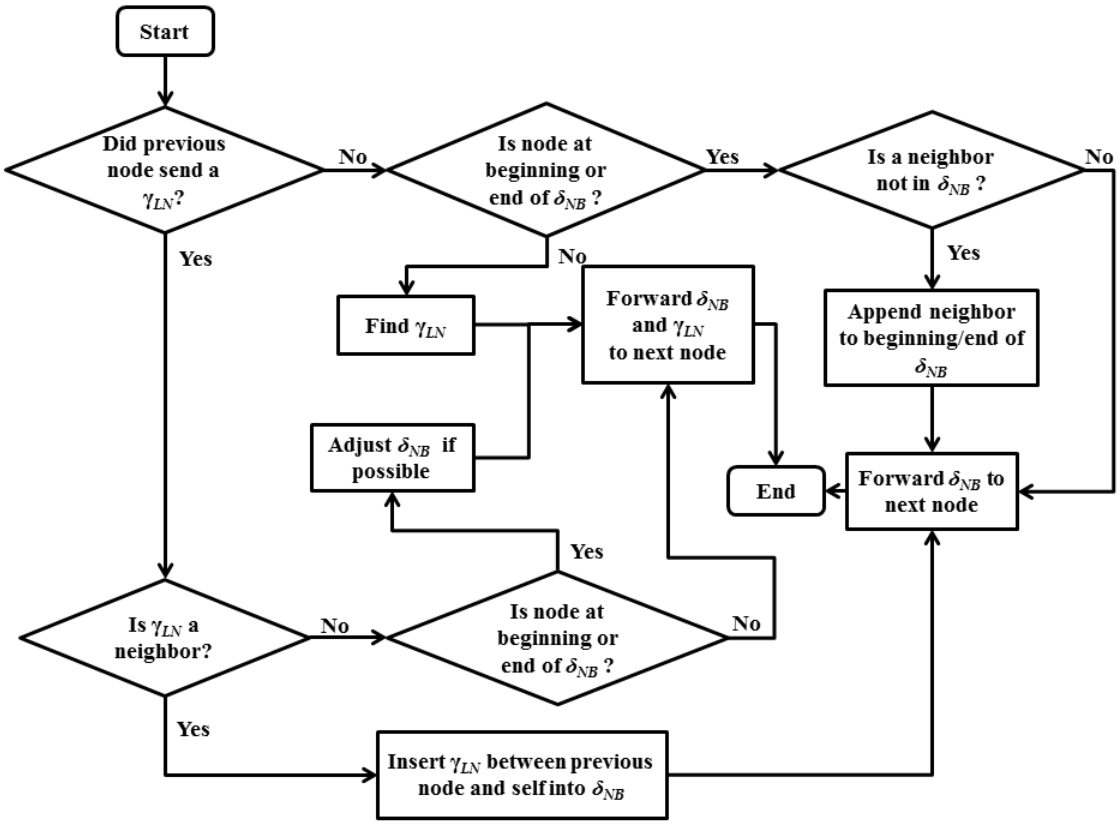

Figure 14. Flow chart illustrating the sequence correction check. Each node conducts this check when it receives δ_{NB} from the previous node in the sequence.

D. PROPOSED HYBRID TIME SYNCHRONIZATION PROTOCOL

In Chapter II, we discussed the merits of the receiver-to-receiver paradigm introduced by RBS and the mean-squared error of R4Syn's relative skew and offset estimators. We also discussed the increased performance of RSP over FTSP for synchronizing one-hop neighbors via broadcast messages. Based on these concepts, in this section, we propose a new R4Syn-RSP hybrid protocol using δ_{NB} generated by the network.

We first provide the motivation for creating a hybrid protocol vice solely relying on RSP or R4Syn. By broadcasting timestamp messages according to δ_{NB} and generating a matrix of timestamps T for each node, we propose a novel method that re-defines T to determine the master reference node γ_{MR} to which all other nodes attempt to synchronize.

Next, we use the timestamps in *T* to form matrices of the relative parameters of each node so that every node has the ability to reference another node's relative parameters for multi-hop R4Syn synchronization. Finally, we present a flow chart that describes how each node determines the method of synchronization, and if necessary, broadcasts additional timing messages to synchronize other neighboring nodes.

1. Motivation for a Hybrid Protocol

Although the proposed broadcast sequencing scheme may address the sequence and collision avoidance problems related to R4Syn, there still exist topologies where R4Syn falls short. We provide motivation for incorporating RSP and forming a hybrid protocol using the topology in Figure 15.

If we attempt to apply broadcast sequencing and *only* R4Syn to the topology in Figure 15, the need for an additional protocol becomes obvious. According to the proposed broadcast sequencing algorithm, the network generates a δ_{NB} of $\{4,5,3,2,7,9,10,11\}$. We assume that a sufficient number of timestamps have been received by each node and are ready to synchronize. For illustrative purposes, say Node 7 is determined to be the master reference node γ_{MR} because it has the most number of neighbors. While Nodes 1, 3, and 10 have timestamp pairs with Node 7 to synchronize their clocks via 1-hop R4Syn, the other nodes are left without any timestamp pairs with Node 7. For this very reason, we require another protocol to assist this time synchronization scheme.

We propose using RSP for nodes that cannot be synchronized directly via R4Syn due to RSP's level of precision of less than half of a microsecond. Given the same topology example in Figure 15, Node 7 broadcasts two additional messages to its one-hop neighbors since none of them can synchronize via R4Syn. For nodes that can synchronize via R4Syn, they adjust their clocks first and then synchronize any unsynchronized neighbors via RSP. For the same example, Node 10 synchronizes its clock to Node 7's clock using the R4Syn parameters and then synchronize Node 11 via RSP by sending two timestamp messages to Node 11.

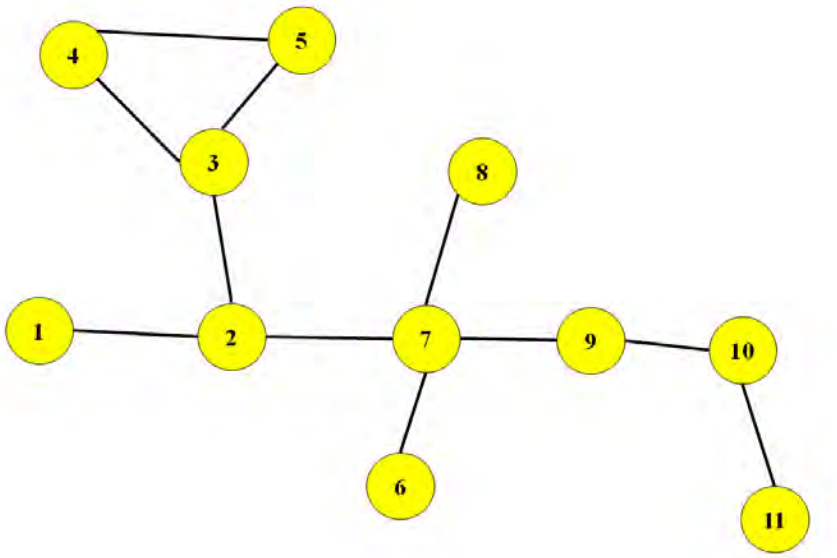

Figure 15. Topology illustrating the need for a hybrid protocol based on broadcast sequencing. Some sparse networks like this cannot conduct accurate network-wide synchronization using only one protocol.

2. Determination of the Master Reference Node

When it is a node's turn to broadcast, it forwards all previously received timestamp information to its neighbors. Receiving nodes simply ignore duplicate information or update their tables as information is received. After a sufficient number of cycles, each node determines the master reference node γ_{MR} by analyzing their timestamp information to find the node that has the most number of timestamp pairs with other nodes.

We consider one cycle as a round-trip sequence of nodes (i.e., nodes transmit messages per the order in δ_{NB} and δ_{NB} reversed). For example, δ_{NB} for the topology in Figure 15 was {4, 5, 3, 2, 7, 9, 10, 11}, so a one-cycle sequence is {4, 5, 3, 2, 7, 9, 10, 11, 10, 9, 7, 2, 3, 5}. To illustrate this idea, we consider the network topology in Figure 8 and re-label the nodes as in Figure 16.

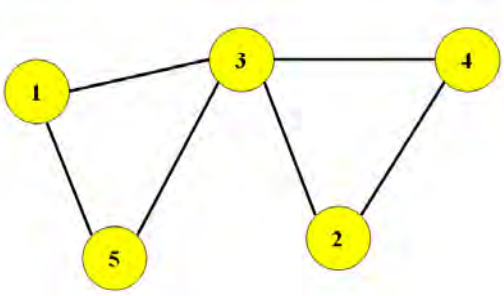

Figure 16. Topology illustrating the re-defintion of *T*. Each node eventually forms the same matrix of timestamps. *T* is used to determine the master reference node and calculate relative clock parameters.

The network determines δ_{NB} to be $\{1,5,3,2,4\}$, which means that all five nodes broadcast during this sequence. We extend the number of broadcasts *M* according to our definition of one cycle so that the final sequence for one cycle is $\{1,5,3,2,4,2,3,5\}$, which means that there is a total of $M = 8$ broadcasts in one cycle for this particular case. In general, for a network of *N* nodes and a δ_{NB} of *w* nodes,

$$M = \begin{cases} 2w-2, & N \geq 4 \\ 4, & N = 3 \end{cases} \tag{17}$$

where *w* is the number of nodes in δ_{NB}, or the length of δ_{NB}. If $N = 2$, we recommend that the pair synchronize using RSP or TPSN. Our research primarily focuses on networks of three or more nodes.

Using the topology in Figure 16 and the resulting timestamps from one cycle, we transform $t_{i,j}^{k}$ of Equation (4) and re-define *T* as

$$T = \begin{bmatrix} * & t_{1,2}^{5} & t_{1,3}^{3} & - & - & - & t_{1,7}^{3} & t_{1,8}^{5} \\ - & - & t_{2,3}^{3} & * & t_{2,5}^{4} & * & t_{2,7}^{3} & - \\ t_{3,1}^{1} & t_{3,2}^{5} & * & t_{3,4}^{2} & t_{3,5}^{4} & t_{3,6}^{2} & * & t_{3,8}^{5} \\ - & - & t_{4,3}^{3} & t_{4,4}^{2} & * & t_{4,6}^{1} & t_{4,7}^{3} & - \\ t_{5,1}^{1} & * & t_{5,3}^{3} & - & - & - & t_{5,7}^{3} & * \end{bmatrix} \tag{18}$$

where $t_{R,M}^{S}$ refers to the reception timestamp at node *R* from the M^{th} broadcast of the cycle sent by node *S*. By observation, the subscript *R,M* also denotes a timestamp's location in the matrix (R^{th} row, M^{th} column). An asterisk " * " denotes the broadcasting node, and a

hyphen " – " denotes that no timestamps were received from the M^{th} broadcast; they are only used in this example to assist the reader of easily identifying the broadcasting node per δ_{NB}.

Given the reception timestamp matrix, each node can determine γ_{MR}. In the example, it is clear that Node 3 (row 3) has the largest number of reception timestamps and can form the most number of timestamp pairs with other nodes and, thus, becomes γ_{MR}. If there were more than one node that could be γ_{MR} due to equal number of reception timestamps, the node with the smallest ID is chosen to be γ_{MR}. So if there was a tie between Nodes 2 and 3, for example, Node 2 is declared as γ_{MR}.

3. Relative Skew and Offset

After every cycle, nodes use their reception timestamp matrix T to calculate their relative skew and offset to γ_{MR} via the estimators in [11]. Given the timestamp matrix from Equation (18) where Node 3 is γ_{MR}, Node 1 can use timestamp pairs $(t_{1,2}^5, t_{3,2}^5)$ and $(t_{1,8}^5, t_{3,8}^5)$, Node 2 can use timestamp pair $(t_{2,5}^4, t_{3,5}^4)$, Node 4 can use timestamp pairs $(t_{4,4}^2, t_{3,4}^2)$ and $(t_{4,6}^1, t_{3,6}^2)$, and Node 5 can use timestamp pair $(t_{5,1}^1, t_{3,1}^1)$. With these timestamp pairs, nodes form a $N \times N$ matrix of relative clock skews f_R and a $N \times N$ matrix of relative clock offsets O_R using Equations (5) and (6) as

$$f_R = \begin{bmatrix} 1 & f_{1,2} & f_{1,3} & \cdots & f_{1,\mathrm{N}} \\ f_{2,1} & 1 & f_{2,3} & \ddots & f_{2,\mathrm{N}} \\ f_{3,1} & f_{3,2} & 1 & \ddots & f_{3,\mathrm{N}} \\ \vdots & \ddots & \ddots & 1 & \vdots \\ f_{\mathrm{N},1} & f_{\mathrm{N},2} & f_{\mathrm{N},3} & \cdots & 1 \end{bmatrix} \tag{19}$$

and

$$\mathrm{O}_R = \begin{bmatrix} 0 & \mathrm{O}_{1,2} & \mathrm{O}_{1,3} & \cdots & \mathrm{O}_{1,\mathrm{N}} \\ \mathrm{O}_{2,1} & 0 & \mathrm{O}_{2,3} & \ddots & \mathrm{O}_{2,\mathrm{N}} \\ \mathrm{O}_{3,1} & \mathrm{O}_{3,2} & 0 & \ddots & \mathrm{O}_{3,\mathrm{N}} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ \mathrm{O}_{\mathrm{N},1} & \mathrm{O}_{\mathrm{N},2} & \mathrm{O}_{\mathrm{N},3} & \cdots & 0 \end{bmatrix}, \tag{20}$$

where the elements $f_{x,y}$ and $\mathrm{O}_{x,y}$ are denoted as Node *x*'s relative skew/offset to Node *y*. Given the example topology in Figure 16, Node 1 can adjust its clock and synchronize to Node 3 by using Equation (1) such that $C_3(t) = f_{1,3} \times C_1(t) + \mathrm{O}_{1,3}$. All other nodes synchronize to Node 3 via the same method using f_R and O_R.

Equations (19) and (20) are also needed for multi-hop synchronization. To illustrate a topology requiring multi-hop R4Syn synchronization, we use the example in Figure 17, where the resulting timestamp matrix for one cycle is

$$T = \begin{bmatrix} * & t_{1,2}^5 & t_{1,3}^3 & - & - & - & - & - & - & - & t_{1,11}^3 & t_{1,12}^5 \\ - & - & t_{2,3}^3 & t_{2,4}^4 & * & t_{2,6}^6 & - & t_{2,8}^6 & * & t_{2,10}^4 & t_{2,11}^3 & - \\ t_{3,1}^1 & t_{3,2}^5 & * & t_{3,4}^4 & t_{3,5}^2 & - & - & - & t_{3,9}^2 & t_{3,10}^4 & * & t_{3,12}^5 \\ - & - & t_{4,3}^3 & * & t_{4,5}^2 & - & - & - & t_{4,9}^2 & * & t_{4,11}^3 & - \\ t_{5,1}^1 & * & t_{5,3}^3 & - & - & - & - & - & - & - & t_{5,11}^3 & * \\ - & - & - & - & t_{6,5}^2 & * & t_{6,7}^7 & * & t_{6,9}^2 & - & - & - \\ - & - & - & - & - & t_{7,6}^6 & * & t_{7,8}^6 & - & - & - & - \end{bmatrix}. \quad (21)$$

With Node 3 as γ_{MR}, Node 6 can synchronize to Node 3 using timestamp pairs $(t_{3,5}^2, t_{6,5}^2)$ and $(t_{3,9}^2, t_{6,9}^2)$; although, it is two hops away from Node 3. While Node 6 is two link-hops away from Node 3, it is only one R4Syn-hop from Node 3 because it has timestamp pairs with γ_{MR}. In the case of Node 7, it does not have any timestamp pairs with γ_{MR} but instead with Node 2. Because Node 2 has timestamp pairs with γ_{MR}, Node 7 is two R4Syn-hops from γ_{MR}. Consequently, Node 7 uses the multi-hop R4Syn parameter estimators presented in Chapter II (see Equations (7) and (8)) to adjust its clock to Node 3's. Thus, Node 7 synchronizes to Node 3 by

$$C_3(t) = f_{7,3} \times C_7(t) + \mathrm{O}_{7,3} \quad (22)$$

where

$$f_{7,3} = f_{7,2} \times f_{2,3}, \quad (23)$$

$$\mathrm{O}_{7,3} = f_{2,3} \times \mathrm{O}_{7,2} + \mathrm{O}_{2,3}, \quad (24)$$

and the relative parameters $f_{7,2}, f_{2,3}, \mathrm{O}_{7,2}$ and $\mathrm{O}_{2,3}$ are extracted from f_R and O_R.

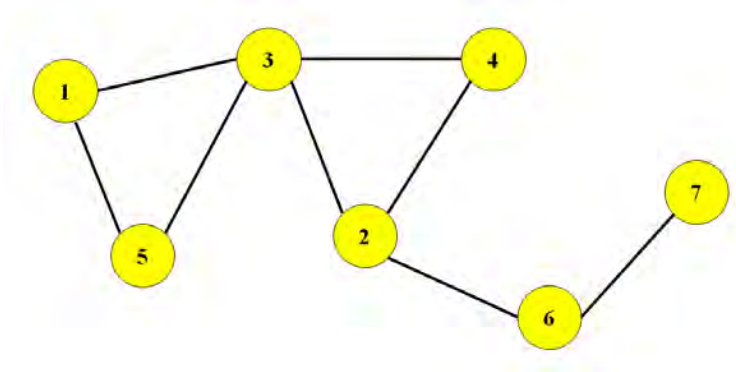

Figure 17. Topology illustrating multi-hop R4Syn synchronization. Nodes use T to determine if they can synchronize directly to the master reference node's clock and synchronize neighbors if necessary.

4. Synchronization Method Flow Chart

Nodes also use the reception timestamp matrix to determine if any of their neighbor nodes require synchronization via RSP by simply examining whether or not their neighbors can synchronize via R4Syn. The method that each node uses to synchronize with the rest of the network as described in the previous sections can be represented by the flow chart displayed in Figure 18. Upon receiving enough timestamp information, each node determines the master reference node γ_{MR} and calculates its relative parameters. If the node can synchronize to γ_{MR} directly via R4Syn, it adjusts its clock using relative parameters. Next, the node checks its timestamp matrix to see if there are any neighbor nodes that cannot synchronize to γ_{MR} via R4Syn. If there are unsynchronized neighbors, the node synchronizes them via RSP using two more timing messages.

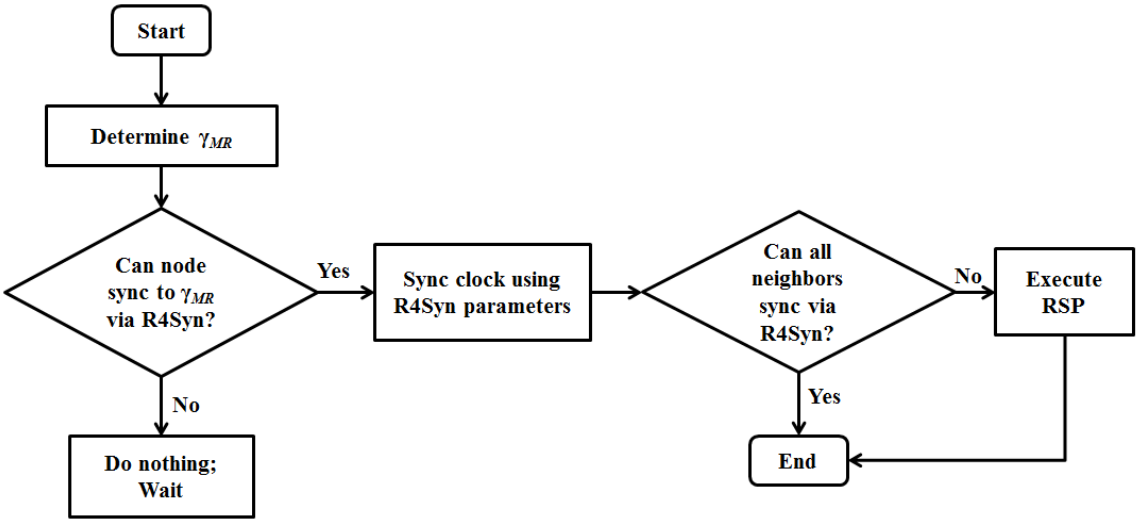

Figure 18. Flow chart illustrating the R4Syn-RSP hybrid time synchronization protocol. R4Syn is the primary protocol of synchronization. RSP is secondary.

In this chapter, we proposed a new hybrid time synchronization scheme. First, we discussed our overall scheme by which an unsynchronized network of ad hoc wireless sensor nodes becomes synchronized through three major steps: determining the most node-inclusive broadcast sequence δ_{NB}, conducting a sequence correction check, and synchronizing nodes via RSP or R4Syn. Starting with δ_{NB} determination, we first presented the concept behind broadcast sequencing and then discussed the procedures that nodes conduct in order to determine their sequences. Next, we demonstrated the need for a sequence correction check for certain topologies and provided additional procedures to adjust the final sequence that make it more inclusive. Finally, we proposed the hybrid R4Syn-RSP scheme used to synchronize the network. In the next chapter, we evaluate the performance of our proposed time synchronization scheme through a series of simulations.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SIMULATION

In this chapter, we evaluate the performance of our proposed hybrid scheme for wireless network time synchronization via MATLAB simulations. We begin by examining and simulating the R4Syn skew and offset estimators in order to validate results in [11]. Next, we explore the performance of the proposed algorithm for determining the network broadcast sequence δ_{NB} through a series of different network simulations. Finally, we apply δ_{NB} to synchronize networks using the proposed hybrid time synchronization scheme. All source code for the simulations in this chapter is included in Appendix B.

A. R4SYN ESTIMATORS

In this section, we conduct MATLAB simulations to validate the effectiveness and accuracy of the R4Syn relative skew and offset estimators proposed in [11] so that the proposed broadcast sequence δ_{NB} can be applied to the new hybrid time synchronization scheme.

1. Simulation Model

We begin by simulating the performance of the estimators via Monte Carlo simulations. The basic simulation scenario is presented in Figures 5 and 19. The simulation places four nodes within communications range of each other. Node 1 begins with a relative offset and relative skew of 0 seconds and 1, respectively, while all other nodes begin with a uniformly random relative offset in the range of [–10, 10] seconds and a uniformly random frequency in the range of [0.75, 1.25]. Propagation times are given by

$$t_{prop} = \frac{d_{x,y}}{c} \tag{25}$$

where $c = 3 \times 10^8$ m/s (speed of light) and $d_{x,y}$ is the Euclidean distance between Nodes x and y. Upon receiving a beacon, each node's processing time t_{proc} is modeled as a

uniformly random variable in the range of [100, 500] milliseconds, which includes all sources of delay (e.g., send/receive, access, transmission/reception).

2. Results

We simulate each node transmitting their beacon upon receiving a beacon from the previous node in sequential order (i.e., 1, 2, 3, 4) and repeating this cycle until 100 timestamp messages are generated for each node. The effectiveness of the relative skew and offset estimators (f_{ML} and O_{ML} , respectively) from Chapter II is measured by calculating the mean-squared error ε_{MS} and the Cramer-Rao lower bound (CRLB) β_{CR} at every tenth message. We denote the mean-squared error of the relative skew estimator as

$$\varepsilon_{MS}(f_{ML}) = \mathrm{E}[(f_{ML} - f_A)^2] \tag{26}$$

and the mean-squared error of the relative offset estimator as

$$\varepsilon_{MS}(\mathrm{O}_{ML}) = \mathrm{E}[(\mathrm{O}_{ML} - \mathrm{O}_A)^2] \tag{27}$$

where f_A and O_A are the actual relative skew and actual relative offset, respectively, that are randomly assigned to a given node as mentioned in the previous section. Recall the definition of timestamp samples (u_k, v_k) and broadcast messages I from Chapter II that are used to calculate f_{ML} and O_{ML} . We denote the CRLB of the relative skew estimator as [11]

$$\beta_{CR}(f_{ML}) \geq \frac{I\sigma^2}{I\sum_{k=1}^{I} v_k^2 - \left(\sum_{k=1}^{I} v_k\right)^2} \tag{28}$$

and the CRLB of the relative offset estimator as [11]

$$\beta_{CR}(\mathrm{O}_{ML}) \geq \frac{\sigma^2 \sum_{k=1}^{I} v_k^2}{I\sum_{k=1}^{I} v_k^2 - \left(\sum_{k=1}^{I} v_k\right)^2} \tag{29}$$

where σ^2 is the variance; $\sigma^2 = 1$ microsecond for this simulation.

We calculate the mean-squared error and the CRLB of the skew and offset estimators after every tenth message, repeat the experiment 1000 times and take the

average. For this one-hop network scenario, we present results in Figures 20 and 21 that match closely to the results from [11]. The two figures demonstrate that ε_{MS} of the estimators decreases and approaches β_{CR} as the number of messages increases.

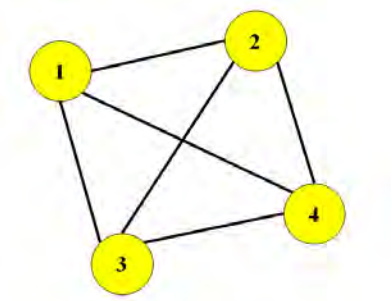

Figure 19. Simulation scenario where four nodes can all communicate with one another.

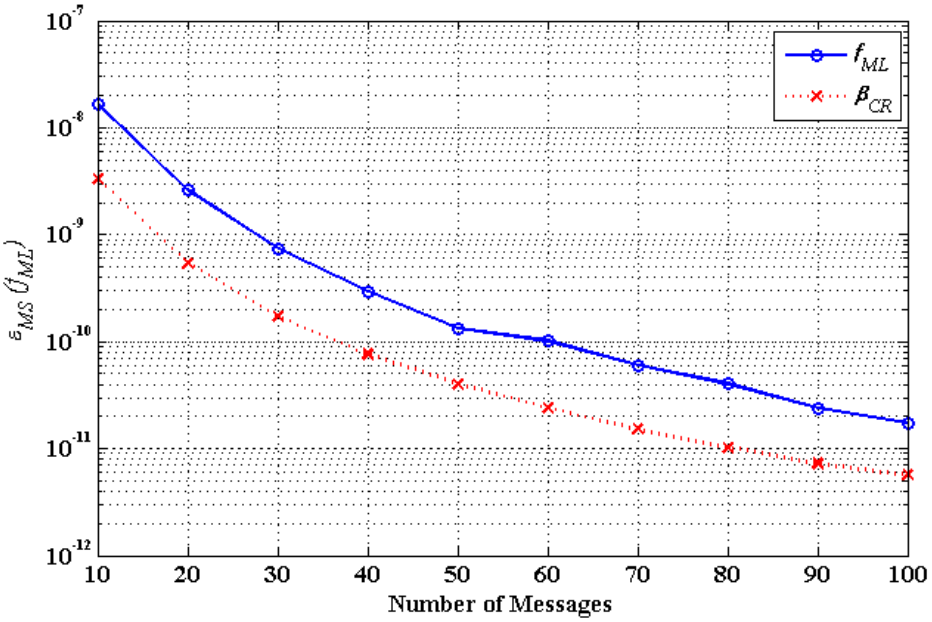

Figure 20. Mean-squared error of skew estimation versus number of messages. As we increase the number of messages, the mean-squared error of the relative skew estimator improves and approaches the CRLB.

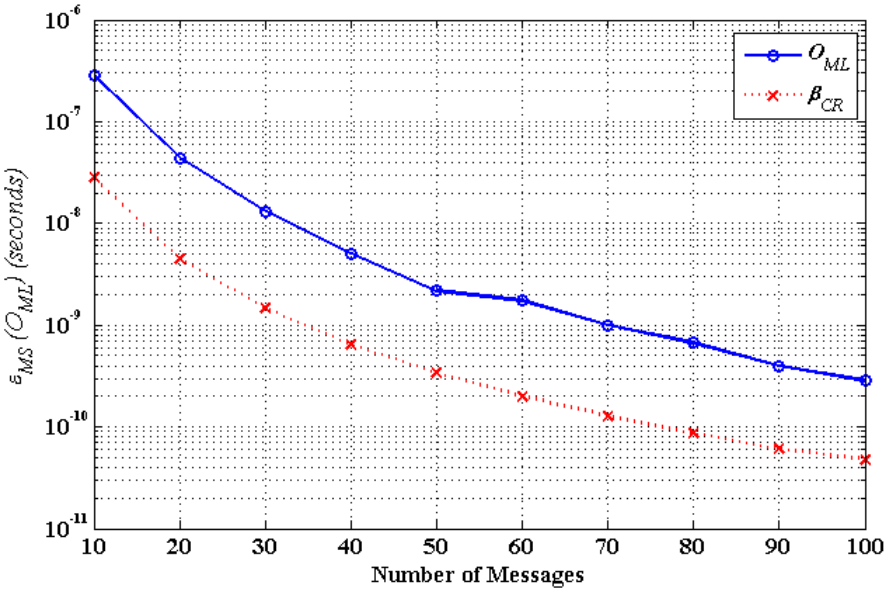

Figure 21. Mean-squared error of offset estimation versus number of messages. As we increase the number of messages, the mean-squared error of the relative offset estimator improves and approaches the CRLB.

We continue the validation by simulating a multi-hop network scenario as shown in Figure 22, where numbered nodes calculate their local one-hop synchronization parameters and perform multi-hop synchronization on demand, similar to the multi-hop network in [11]. For example, Node 4 is three link-hops away from Node 1; thus, it must receive local synchronization parameters from Nodes 1, 2, and 3 to perform multi-hop synchronization.

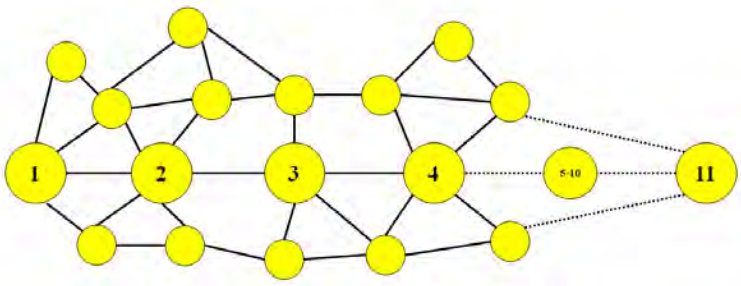

Figure 22. Multi-hop network scenario

The results in Figure 23 show that the mean-squared error for the multi-hop skew estimator improved as the number of messages increased even with a higher number of hop values, as expected. The simulation results also validate that the mean-squared error is kept at an acceptable range for a small number of messages as well as a large number of hops.

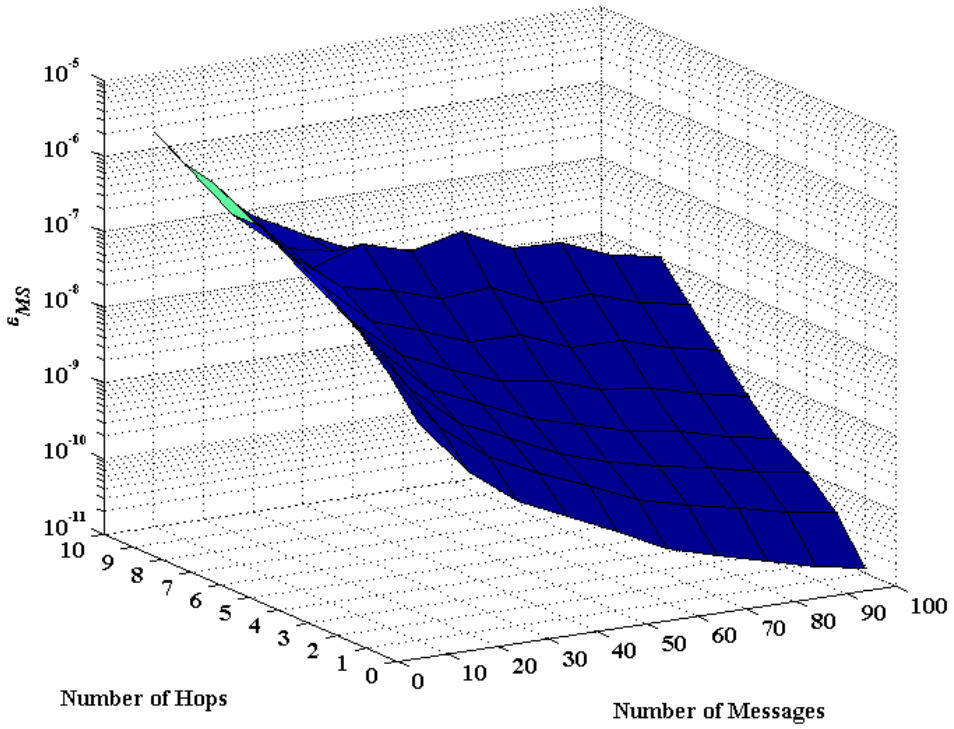

Figure 23. Mean-squared error of multi-hop skew estimation versus number of messages. As we increase the number of messages and hops, the mean-squared error of the multi-hop skew estimator is kept at an acceptable range for time synchronization.

Through our simulation results, we validate how the proposed estimators' mean-squared errors decrease and approach the CRLB as we increase the number the timestamp messages for the one-hop scenario and that the mean-squared errors provide an acceptable range of accuracy regardless of the number of hops or messages. This leads us to affirm that the estimators are, in fact, an effective and accurate tool for time synchronization.

B. BROADCAST SEQUENCING PERFORMANCE

In this section, we discuss the simulation used to evaluate the proposed algorithm for determining δ_{NB} as well as the sequence correction check. Summaries of our results are shown in Tables 2 and 3.

1. Simulation Model

We must first describe the formation of dense and sparse network topologies in the simulations. Both types of networks start with the first node centered at the origin of a Cartesian grid, and all nodes are assumed to have the same maximum communications range of 100 meters. For dense networks, every subsequent node is randomly deployed within 33-66 meters of the previous node's location. For sparse networks, every subsequent node is randomly deployed within 66-99 meters of the previous node's location. This methodology ensures that every node can communicate with at least one other node and that the network can form a diverse series of topologies for analysis. For dense networks, the average number of neighbors per node was between 60 percent and 88 percent of the total number of nodes in the network, while the average number of neighbors per node for sparse networks was between 8 percent and 32 percent.

We begin by forming a dense network of $N = 5$ nodes that are deployed with the ability to communicate at least with one other node as shown in Figure 24. Let us assume that one-hop neighbor discovery is complete so that all nodes know the IDs of their one-hop neighbors. Say Node 1 starts with the role of γ_{SN} and floods the network to begin node relationship and sequence determinations. Childless nodes determine their sequences first and then pass their sequences, Γ_F , and N_P to their parent nodes as discussed in Chapter III. This continues until γ_{SN} receives all of the information it needs to determine δ_{NB} . We denote the number of nodes included in δ_{NB} before sequence correction as N_{BSC} and measure our algorithm's sequence forming performance before sequence correction as

$$P_{BSC} = \frac{N_{BSC}}{N} \times 100\% \tag{30}$$

where N is the total number of nodes.

After δ_{NB} is determined, we simulate the network adjusting (as necessary) and forwarding δ_{NB} throughout the network per the sequence correction method in Chapter III. We denote the number of nodes included in δ_{NB} after sequence correction as N_{ASC} and measure our algorithm's sequence forming performance after sequence correction as

$$P_{ASC} = \frac{N_{ASC}}{N} \times 100\% . \tag{31}$$

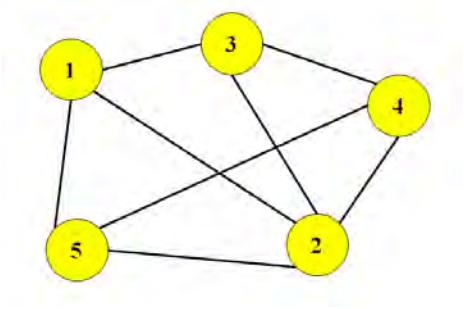

Figure 24. Example topology of a dense network of five nodes.

We distribute the role of γ_{SN} throughout the network for this topology, repeat the process for a total of 100 newly-generated topologies, and, finally, average P_{BSC} and P_{ASC} across the total number of trials to obtain the results provided in the next section. For every new topology that is generated, the location of each node after the deployment of the first node (at the origin) is a new randomly-generated Cartestian grid location; thus, the distances between nodes vary for each topology. We repeat the simulation for dense networks while varying N from 5, 10, 15, 20, 30, 40, to 50 and then do the same process for sparse networks. Intuitively, networks with more nodes will yield more topology variation.

2. Results

A summary of the results for dense networks is displayed in Table 2. Our algorithm for determining δ_{NB} before sequence correction performed significantly well independent of the number of nodes in the networks. The sequence correction check improved the sequence forming performance to or very close to 100 percent of the total number of nodes in the network.

Table 2. Summary of results for dense networks before and after sequence correction. The broadcast sequencing algorithm proved to be effective in including as many nodes as possible for dense networks.

Dense Networks			
Trials = Nodes × Topologies		P_{BSC}	P_{ASC}
Nodes	5	99.28%	99.52%
Topologies	100		
Trials	500		
Nodes	10	99.46%	100.00%
Topologies	100		
Trials	1000		
Nodes	15	99.40%	99.99%
Topologies	100		
Trials	1500		
Nodes	20	99.32%	99.99%
Topologies	100		
Trials	2000		
Nodes	30	99.68%	100.00%
Topologies	100		
Trials	3000		
Nodes	40	99.75%	100.00%
Topologies	100		
Trials	4000		
Nodes	50	99.66%	100.00%
Topologies	100		
Trials	5000		

For sparse networks, the performance of the δ_{NB} algorithm before sequence correction deteriorated as we increased the number of nodes in the network as shown in Table 3; however, the sequence correction algorithm was able to add nodes into the original sequence, increasing the overall percentage to a more acceptable δ_{NB}.

Table 3. Summary of results for sparse networks before and after sequence correction. The sequence correction algorithm proved to be effective in including as many nodes as possible for sparse networks.

Sparse Networks			
Trials = Nodes × Topologies		P_{BSC}	P_{ASC}
Nodes	5	99.08%	100.00%
Topologies	100		
Trials	500		
Nodes	10	94.25%	98.03%
Topologies	100		
Trials	1000		
Nodes	15	88.84%	95.41%
Topologies	100		
Trials	1500		
Nodes	20	82.34%	92.92%
Topologies	100		
Trials	2000		
Nodes	30	72.76%	88.50%
Topologies	100		
Trials	3000		
Nodes	40	67.65%	85.63%
Topologies	100		
Trials	4000		
Nodes	50	60.96%	83.28%
Topologies	100		
Trials	5000		

In this section, we described the methodology behind the simulations for determining δ_{NB} and presented results of the algorithm's performance. The algorithm

performed well for dense networks without a sequence correction check but deteriorated for sparse networks as we increased the number of nodes. The simulations for sparse networks validated the usefulness and need for a sequence correction check for the overall scheme.

C. NETWORK TIME SYNCHRONIZATION USING HYBRID SCHEME

In this section, we evaluate the implementation of δ_{NB} in the proposed hybrid time synchronization scheme for any network topology. We describe the different simulation models used to investigate the effects of increasing hop counts from the master reference node γ_{MR} and the effects of node density on the overall precision of the network's time synchronization. We present results to validate the performance of our proposed hybrid time synchronization scheme for any network topology.

1. Effect of Hop Counts on Network Synchronization

We begin evaluating the hybrid time synchronization scheme by exploring the effects of hop counts from γ_{MR} on the overall precision of the network's synchronization. We present the simulation model and results in this section.

a. Simulation Model

We form the simulation model by creating a linear network of 22 nodes, as displayed in Figure 25. The broadcast sequencing method outputs a final sequence of $\{1, 2, 3, ..., 22\}$ and then nodes broadcast in the order of δ_{NB} for 10 cycles, generating their reception timestamp matrix *T* from (18) in Chapter III. We choose 10 cycles in order to produce a sufficient number of messages to demonstrate whether a synchronization trend exists. We force Node 1 to be γ_{MR} so that only odd-numbered nodes can synchronize directly to Node 1 via R4Syn, while even-numbered nodes are synchronized by odd-numbered nodes via RSP. Nodes synchronize their clocks after every tenth cycle, and we obtain each node's clock precision to γ_{MR} as well as the network's average clock precision to γ_{MR}. In this linear network of 22 nodes, there is a 10-hop maximum for R4Syn synchronization.

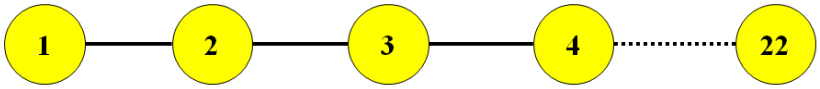

Figure 25. Linear network of 22 nodes.

After observing the precision of the network's time synchronization, we move two of the farthest nodes from γ_{MR} (Nodes 21 and 22) within range of γ_{MR} to form a topology shown in Figure 26, where the maximum R4Syn hop from γ_{MR} is now nine hops. We continue this trend, conducting 100 simulations for each topology, until all nodes are within range of γ_{MR}. The results are detailed in the following subsection.

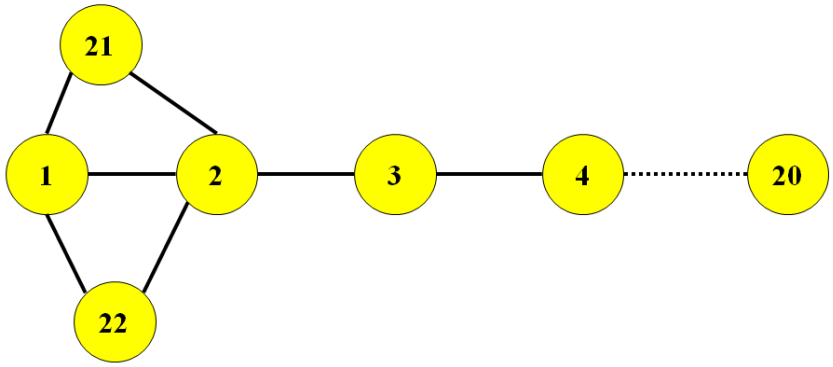

Figure 26. An example of the iterative change in topology used to analyze the effect of hop count on network precision.

b. Results

After conducting 100 simulations for the linear network of 22 nodes, we obtain the results shown in Figure 27. Nodes synchronizing via R4Syn obtain tenths of nanosecond precision to γ_{MR}'s clock, whereas nodes synchronizing via RSP maintain close to 0.5 μs precision to γ_{MR}'s clock . We observe that when a node is a small number of hops away from γ_{MR} for R4Syn synchronization, its clock is more precisely synchronized to γ_{MR}'s clock as shown in Figure 28.

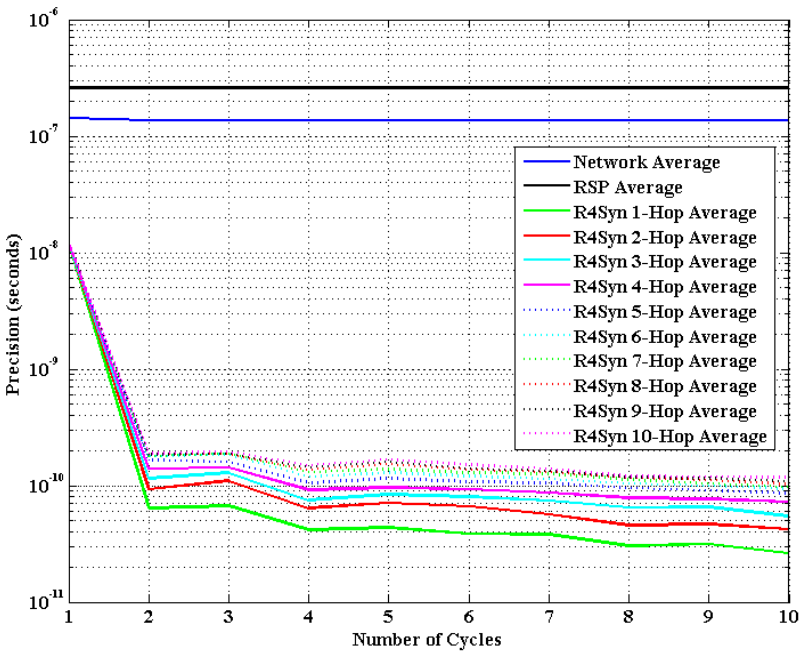

Figure 27. Synchronization results of 100 topologies of a linear network of 22 nodes. Nodes synchronizing via R4Syn obtain tenths of nanosecond precision, whereas nodes synchronizing via RSP maintain a 0.5 μs precision.

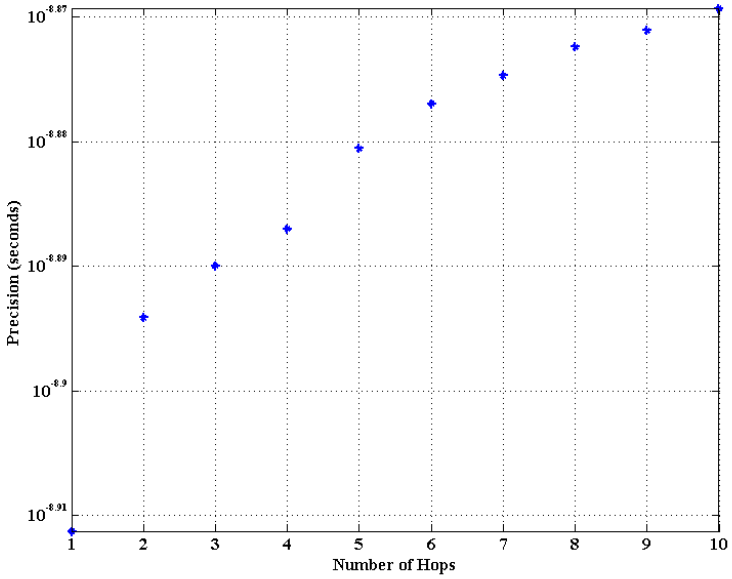

Figure 28. Number of hops versus precision for linear network of 22 nodes. Node precision slightly deteriorates when the number of hops from the master reference node increases for nodes synchronizing via R4Syn.

While we may get near the tenths of nanosecond precision for individual nodes synchronizing via R4Syn, we get the poorest results for the network's precision as a whole as shown in Figure 29. As we changed the topologies by moving the two farthest nodes within range of γ_{MR} and observing the network's clock precision to γ_{MR}, we obtained the results in Figure 29. As expected, the average network synchronization improves when we have a small number of nodes that are multiple hops away from γ_{MR}.

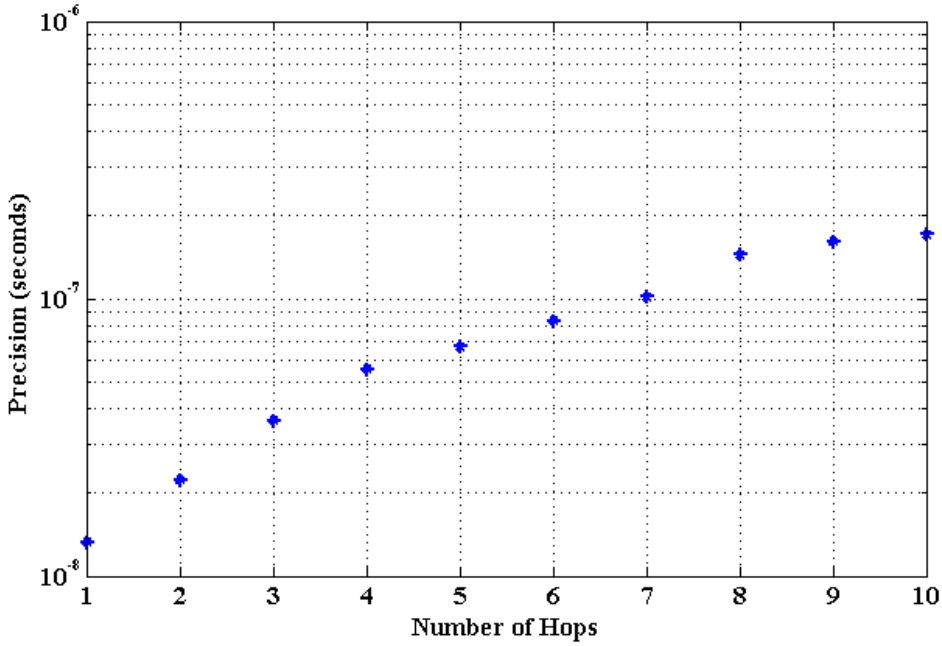

Figure 29. Network synchronization for different topologies. Synchronization improves when less nodes are multiple hops away from the master reference node.

When all nodes are within range of γ_{MR}, we consistently obtain the precision results in the tens of nanoseconds range as displayed in Figure 30. For the network as a whole, synchronization precision is the highest when all nodes are within communications range of one another.

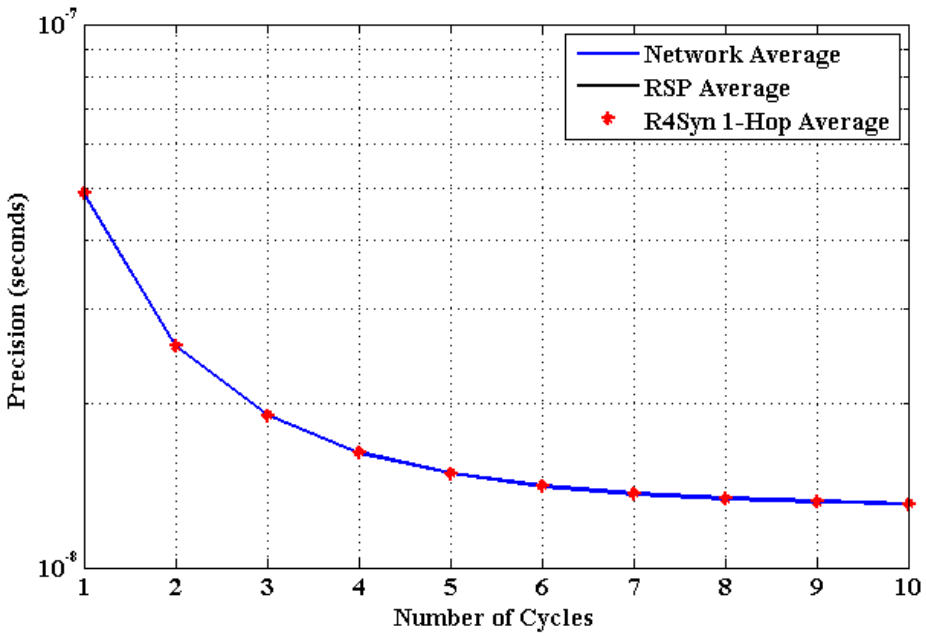

Figure 30. Synchronization results for a fully-connected network where all nodes can communicate with one another. After one cycle, the synchronization precision is 49 nanoseconds. As we increase the number of cycles, synchronization precision improves.

2. Effects of Network Density on Network Synchronization

Next, we explore the effects of network density on the overall precision of the network. Accordingly, we present the network density simulation model and results in this section.

a. Simulation Model

We measure the network density by the average number of neighbors per node for any given topology. We also investigate the performance of our overall time synchronization scheme of determining a network broadcast sequence, performing sequence correction, and then synchronizing via R4Syn or RSP. To simulate this, we generate the same two types of networks (sparse and dense) that we used in the previous section to evaluate δ_{NB}. After the nodes are deployed and discover their one-hop

neighbors, the starter node γ_{SN} is arbitrarily selected to flood the network with a broadcast sequence initiation message. Once determination of δ_{NB} is complete, nodes conduct a sequence correction check and broadcast messages for ten cycles, synchronizing their clocks to γ_{MR} after every tenth cycle. Along the lines of the previous section, we choose ten cycles because a sufficient number of messages are produced to demonstrate whether a synchronization trend exists.

b. Results

We began with sparse networks of 25 nodes and observed the network synchronization statistics for 100 newly-generated topologies. We choose 100 as the number of topologies because it provided a reasonable amount of data without consuming an excessive amount of time to complete simulations. The simulation results in Figure 31 validate that as the number of hops increases, the average precision deteriorates, as expected from the previous section. After one cycle, the network maintains synchronization of less than 0.1 μs precision.

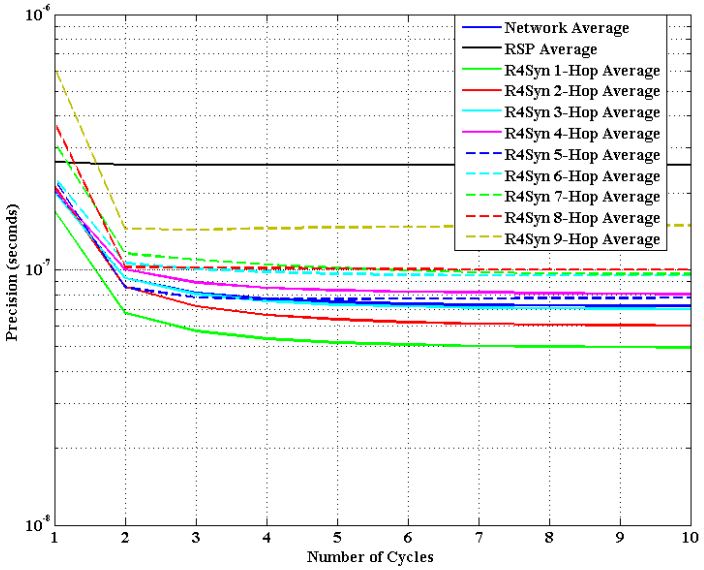

Figure 31. Average synchronization of 100 newly-generated topologies of 25-node sparse networks. After one cycle, the network maintains synchronization of less than 0.1 μs precision.

We increased the number of nodes to 50 and repeated the simulation for sparse networks. The results in Figure 32 show that as the number of nodes in the network doubled, the average network precision slightly deteriorated but is kept at an acceptable range. On average, the network maintains synchronization of less than 0.1 μs precision after about four cycles.

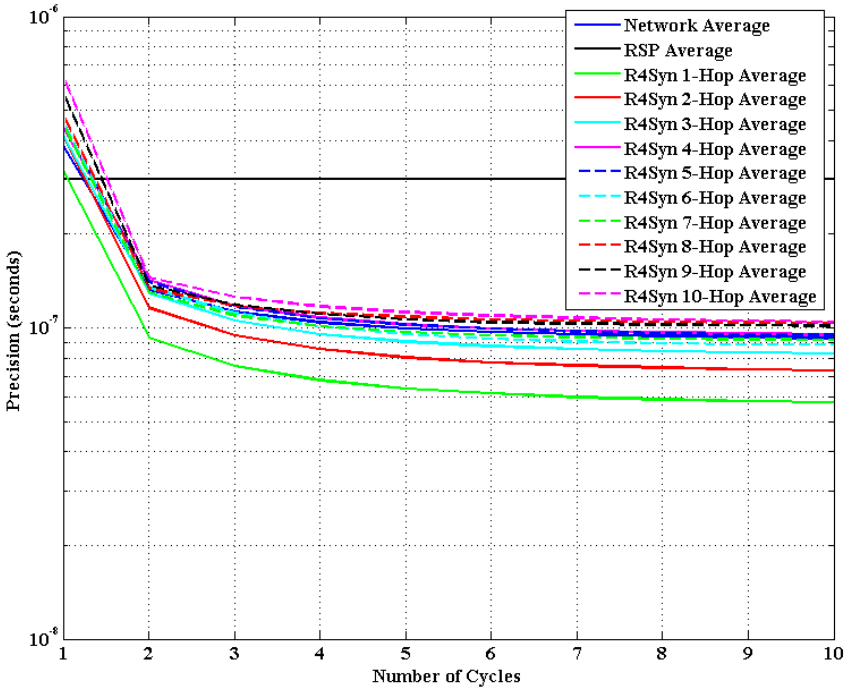

Figure 32. Average synchronization of 100 newly-generated topologies of 50-node sparse networks. After about four cycles, the network maintains synchronization of less than 0.1 μs precision.

We continued our simulations for dense networks of 25 nodes. The simulation results in Figure 33 demonstrate that on average, it takes about four cycles for dense networks to maintain synchronization of less than 0.1 μs precision.

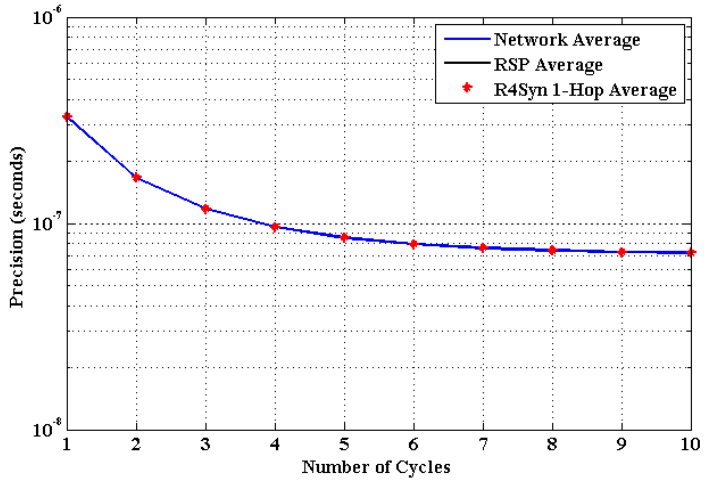

Figure 33. Average synchronization of 100 newly-generated topologies of 25-node dense networks. After about four cycles, the network maintains synchronization of less than 0.1 μs precision.

We increased the number of nodes to 50 and repeated the simulation for dense networks. The results in Figure 34 show that as the number of nodes in the network doubled, the average network precision slightly deteriorated similarly to sparse networks but is still kept at an acceptable range. On average, the network reached synchronization of less than 0.1 μs precision after about ten cycles.

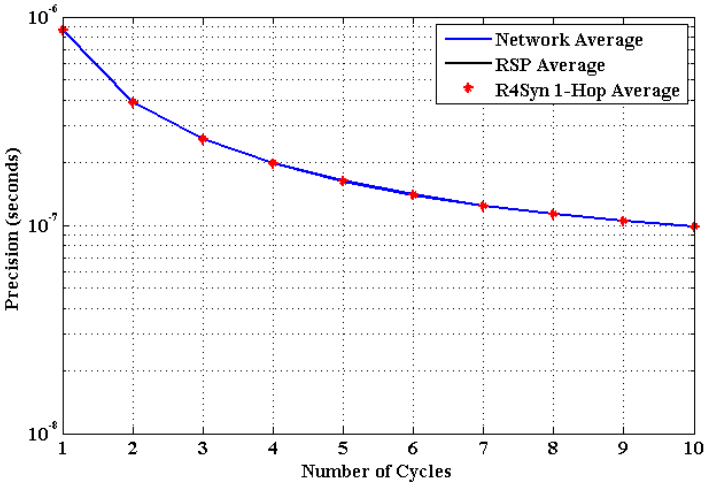

Figure 34. Average synchronization of 100 newly-generated topologies of 50-node dense networks. On average, these networks required at least ten cycles to reach 0.1 μs precision.

Next, we compared the network densities (average number of neighbors per node) for the two types of networks and found that there was little variability in the average network synchronization. The majority of topologies for 25-node sparse networks reached an average synchronization precision of less than 0.1 μs as shown in Figure 35.

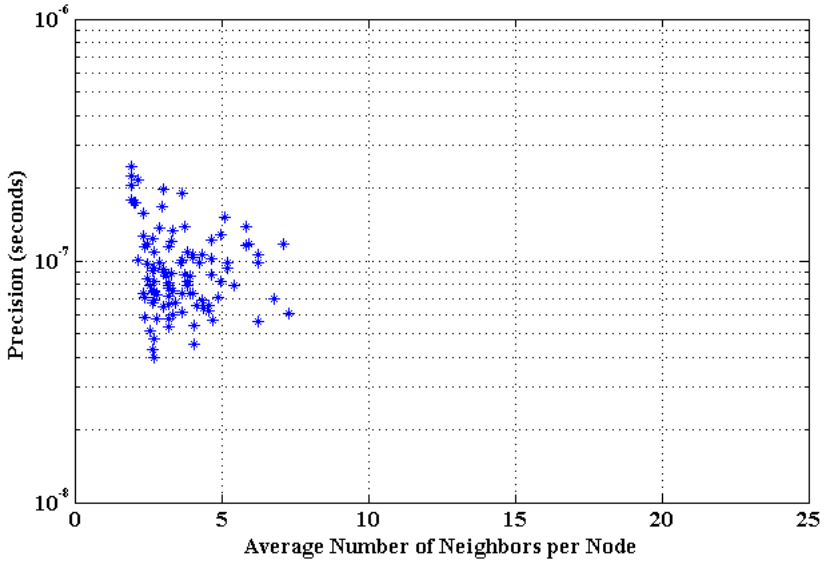

Figure 35. Node density versus average clock precision for sparse networks of 25 nodes. The majority of topologies maintained synchronization of less than 0.1 μs .

When we compared the average synchronization of 25-node sparse networks to 25-node dense networks, we found that the majority of dense topologies reached an average synchronization of around 0.1 μs as shown in Figure 36, which is not a significant difference.

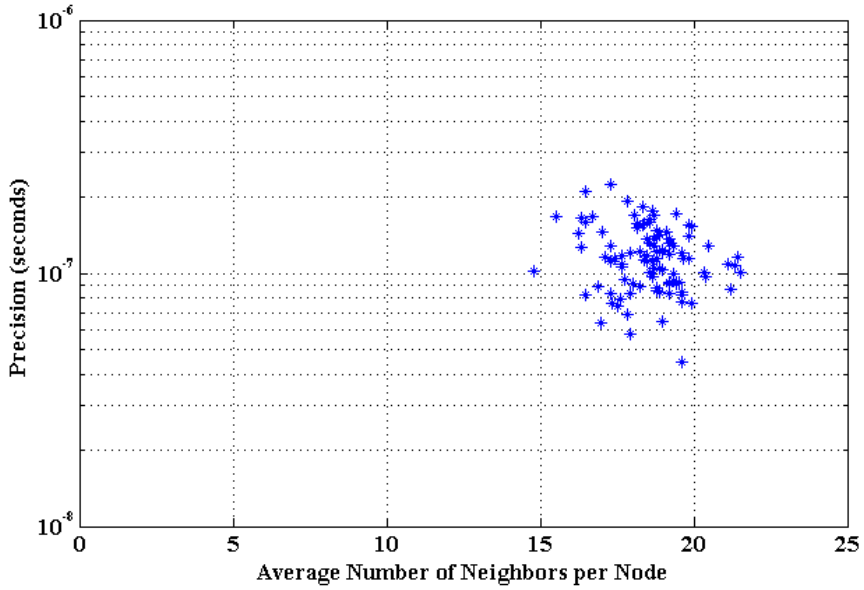

Figure 36. Node density versus average clock precision for dense networks of 25 nodes. The majority of topologies maintained synchronization of around 0.1 μs .

When the number of nodes was increased to 50 for sparse networks, the majority of topologies reached an average synchronization of more than 0.1 μs as shown in Figure 37, which is a decline compared to networks of 25 nodes (see Figure 35).

When the number of nodes was increased to 50 for dense networks, the majority of topologies reached an average synchronization of about 0.2 μs as shown in Figure 38, which is also a decline compared to networks of 25 nodes (see Figure 36).

Although these simulation results demonstrated that a network's time synchronization degrades as the number of nodes increases using the hybrid scheme, the average level of precision for each network was higher than other time synchronization methods regardless of network topology.

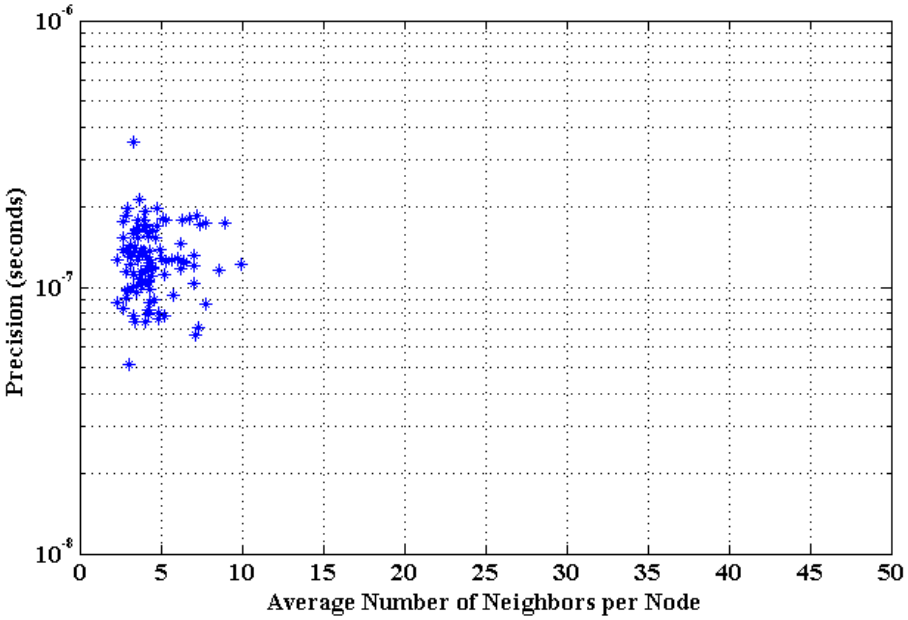

Figure 37. Node density versus average clock precision for sparse networks of 50 nodes. The majority of topologies maintained synchronization of slightly more than 0.1 μs , a decline from 25 node networks.

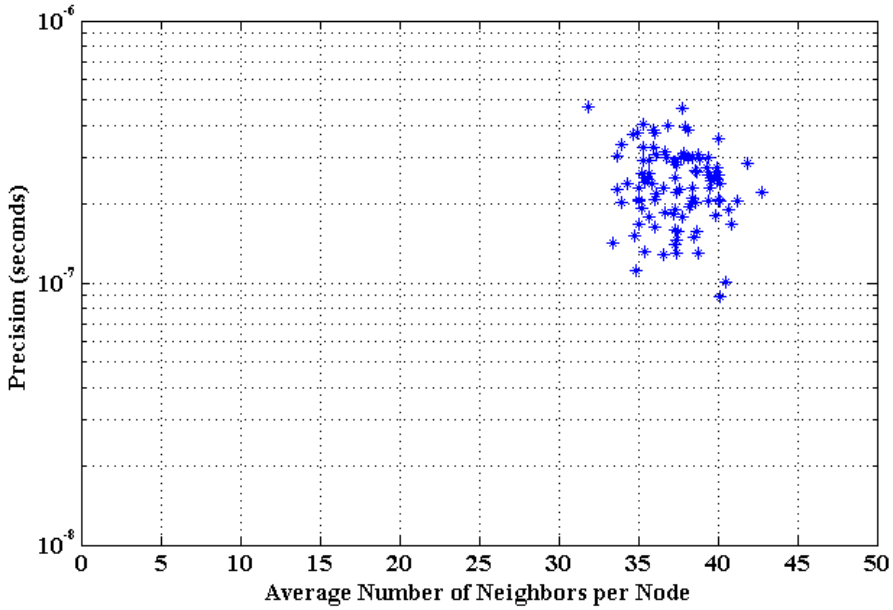

Figure 38. Node density versus average clock precision for dense networks of 50 nodes. The majority of topologies maintained synchronization of about 0.2 μs , a decline from 25 node networks.

In this chapter, MATLAB simulations were conducted to evaluate the performance of our proposed hybrid scheme for wireless network time synchronization. We first validated the R4Syn skew and offset estimators presented in Chapter II. Next, we presented a series of simulations to test the performance of δ_{NB} determination and demonstrate the utility of a sequence correction check. Finally, we explored the performance of the proposed hybrid scheme using broadcast sequencing through another series of simulations. The results demonstrate that our proposed hybrid scheme improves average network synchronization over other existing time synchronization methods regardless of network topology.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

In this thesis, we explored the challenges associated with time synchronization in wireless sensor networks, such as the possibility of collisions and topology changes. A review of related literature revealed that other methods exist for addressing these challenges, to include RSP and R4Syn, which we incorporated in to our proposed hybrid time synchronization scheme. To address the drawbacks of the other methods, a new algorithm for determining a network broadcast sequence for a wireless sensor network was proposed and developed to eliminate the possibility of collisions. Additionally, a model to check and adjust the broadcast sequence for potential errors was presented. Finally, a new hybrid time synchronization scheme of RSP and R4Syn based on the new broadcast sequence developed in this thesis was proposed.

Simulation models were developed to validate the estimators proposed in R4Syn and evaluate the performances of our network broadcast sequencing algorithm and our new hybrid time synchronization scheme. Simulation results demonstrated that our algorithms for determining the broadcast sequence and for the sequence correction were effective in producing a final broadcast sequence with a high percentage of nodes. Simulation results validated that for any topology, the network's time synchronization is more precise using our new hybrid time synchronization scheme than other known time synchronization protocols.

A. SIGNIFICANT CONTRIBUTIONS

Three main contributions were made in this thesis. In order to address the possibility of collisions using R4Syn, a new scheme that determined a specific sequence order and eliminated collisions was presented. Our new algorithms for determining a specific network broadcast sequence and for sequence correction solved the potential collision problem that is inherent in wireless sensor networks.

We introduced a new method of determining a master reference node to synchronize to and performing multi-hop synchronization using an analysis of time stamp

information. This eliminates the need for on-demand synchronization, which leads to more transmissions and a higher probability of collisions.

Finally, we presented a new hybrid time synchronization scheme that approached an average network time synchronization in the nanosecond range for any topology, whereas other known synchronization methods offer time synchronization in the microsecond range. For example, the average network synchronization after one cycle was approximately 49 nanoseconds in a fully-connected network where all nodes are within communications range of one another. For this type of network, each node only has to transmit two broadcasts in one cycle after sequence determination to achieve such a high level of precision. The average network synchronization for multi-hop networks of 25 nodes was approximately 81 nanoseconds after three cycles, which equates to an individual node broadcasting no more than eight messages after sequence determination to achieve that level of precision.

B. RECOMMENDED FUTURE WORK

In this thesis, we explored a new hybrid time synchronization scheme for wireless sensor networks. Simulations verified the validity of our proposed scheme but can be extended to explore the energy usage and consumption during each cycle of broadcasting as well as during computing. These results can be used in conjunction with load balancing techniques to develop optimum power management techniques for energy savings. The simulations in this thesis validated network synchronization for static ad hoc wireless sensor networks in which nodes remain in the same location after deployment. Additional simulations can be undertaken to examine node mobility using our hybrid time synchronization scheme and explore the effects of dynamic topology changes on both the synchronization and the power consumption of a network.

The work in this thesis is all based on simulations. Future implementation on sensor motes (hardware) should be conducted to further investigate the accuracy of our claims and verify our results. The other recommended research areas such as energy savings and node mobility using our hybrid scheme should be compared to other time synchronization methods using sensor motes as well.

APPENDIX A. ADDITIONAL FLOW CHARTS

Additional flow charts used for sequence determination from Chapter III are provided in this appendix.

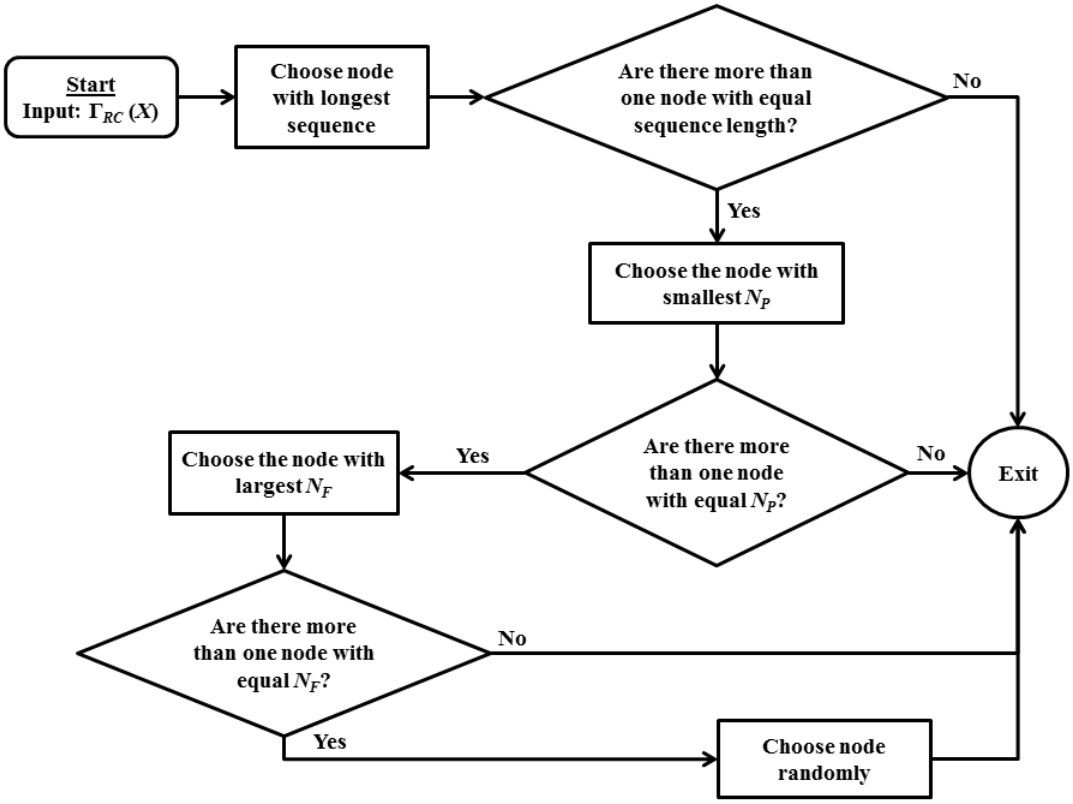

Figure 39. Flow chart to find the child node with the longest sequence.

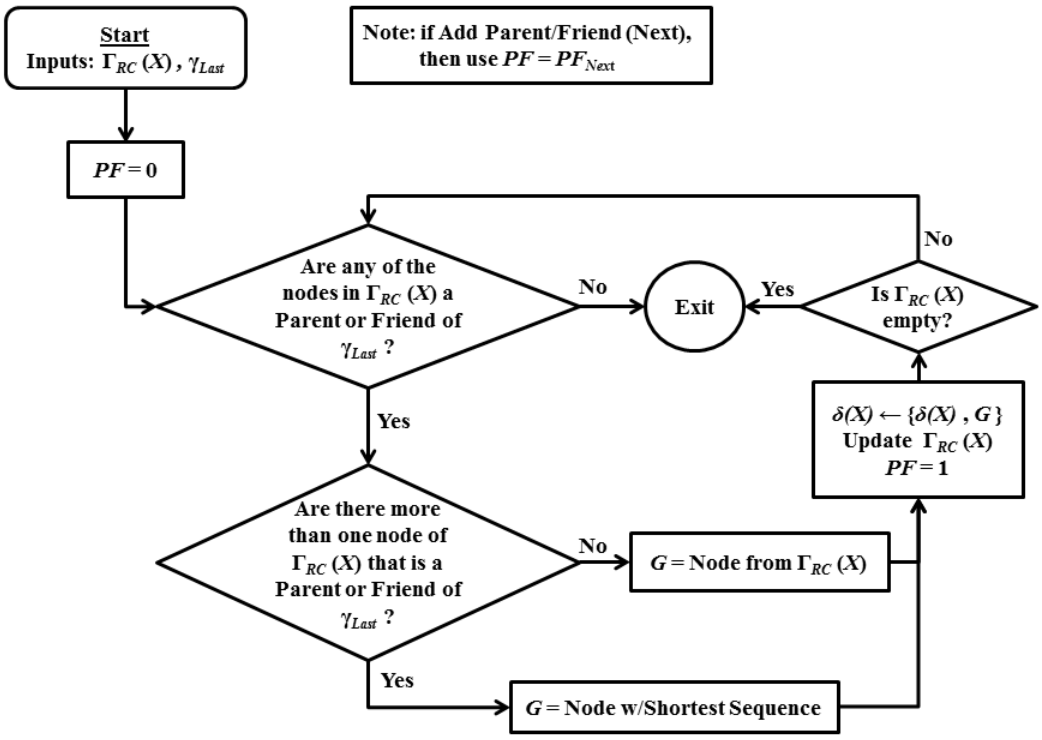

Figure 40. Flow chart to add a parent or friend node to a sequence.

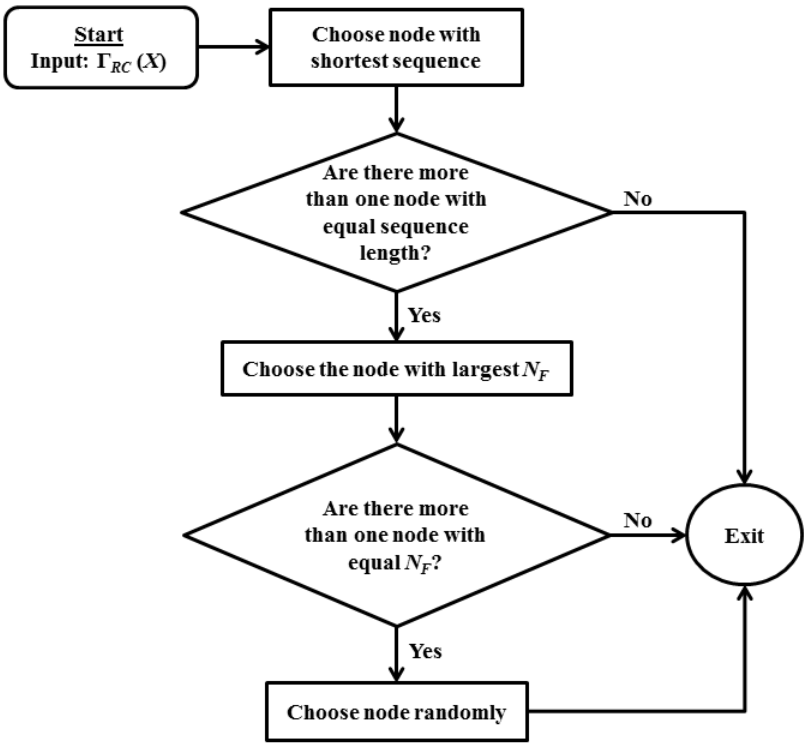

Figure 41. Flow chart to find the node with the shortest sequence.

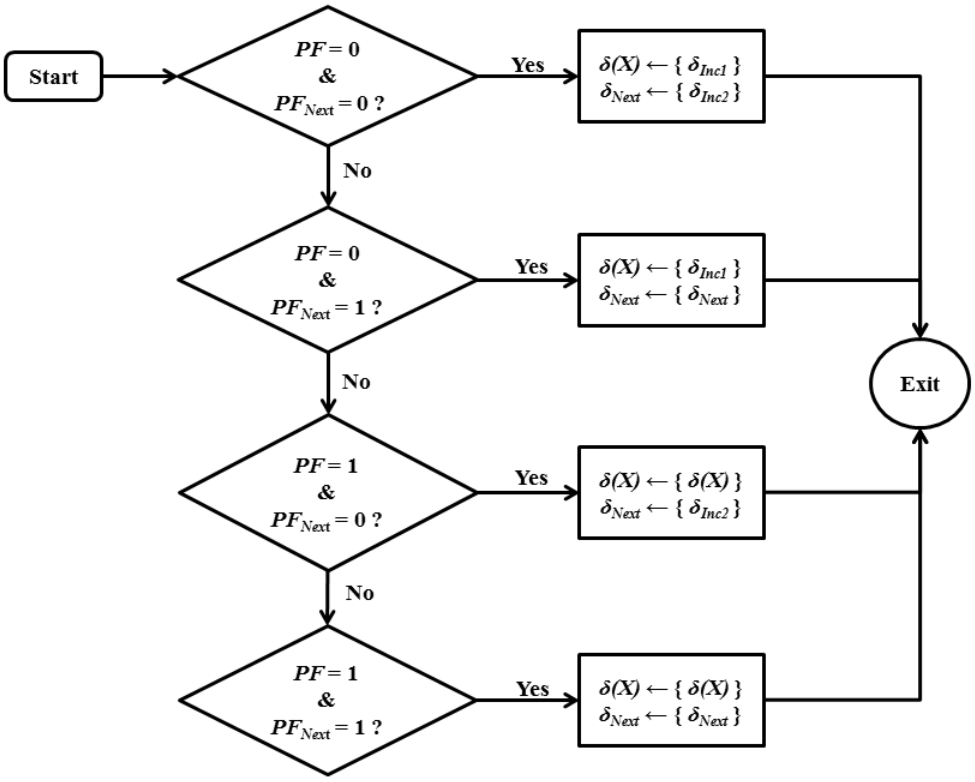

Figure 42. Flow chart updating $\delta(X)$ and δ_{Next}.

To clarify the operation that δ_{Temp} requires in Figures 43–45, let $\delta_{Temp} = \{5,4,3,2,1\}$ and $\delta_{Next} = \{7,8,3,9,6\}$; δ_{Temp} (1:before first similarity with δ_{Next}) refers to the sequence $\{5,4\}$ since the first similarity between δ_{Temp} and δ_{Next} is Node 3, so δ_{Temp} is adjusted to only include $\{5,4\}$.

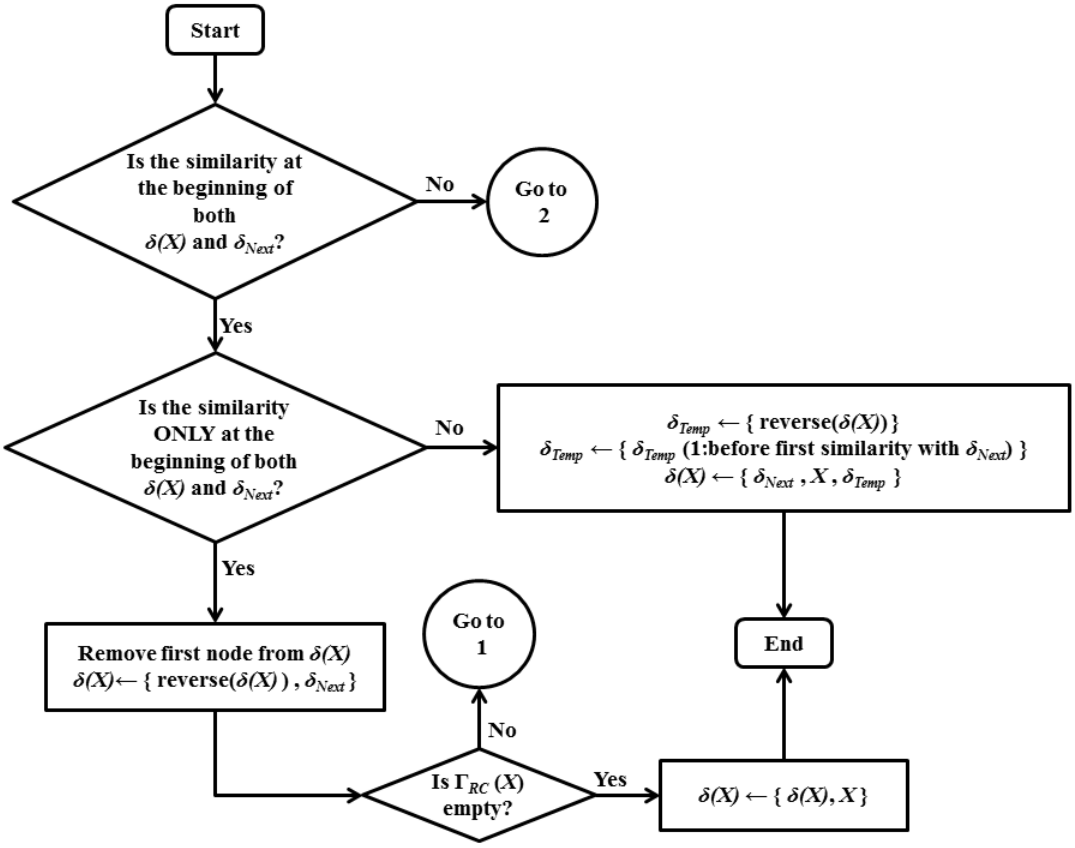

Figure 43. First flow chart adjusting $\delta(X)$ for similarities.

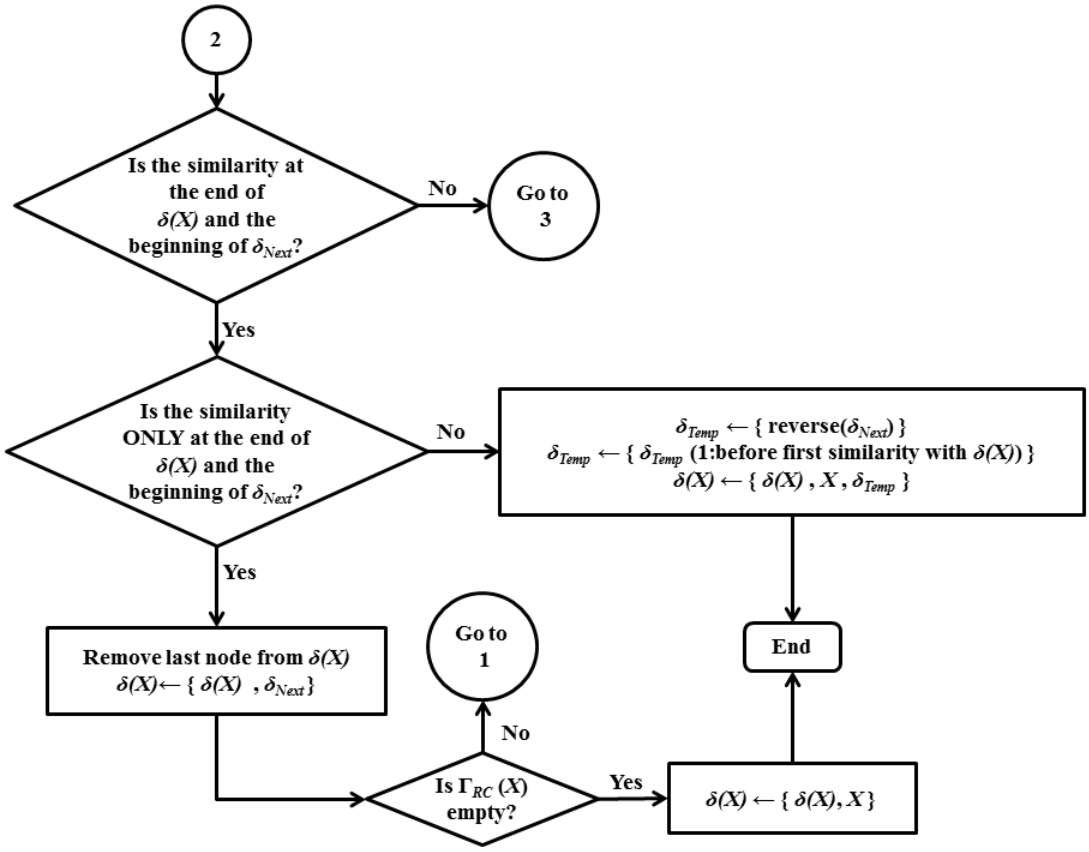

Figure 44. Second flow chart adjusting $\delta(X)$ for similarities.

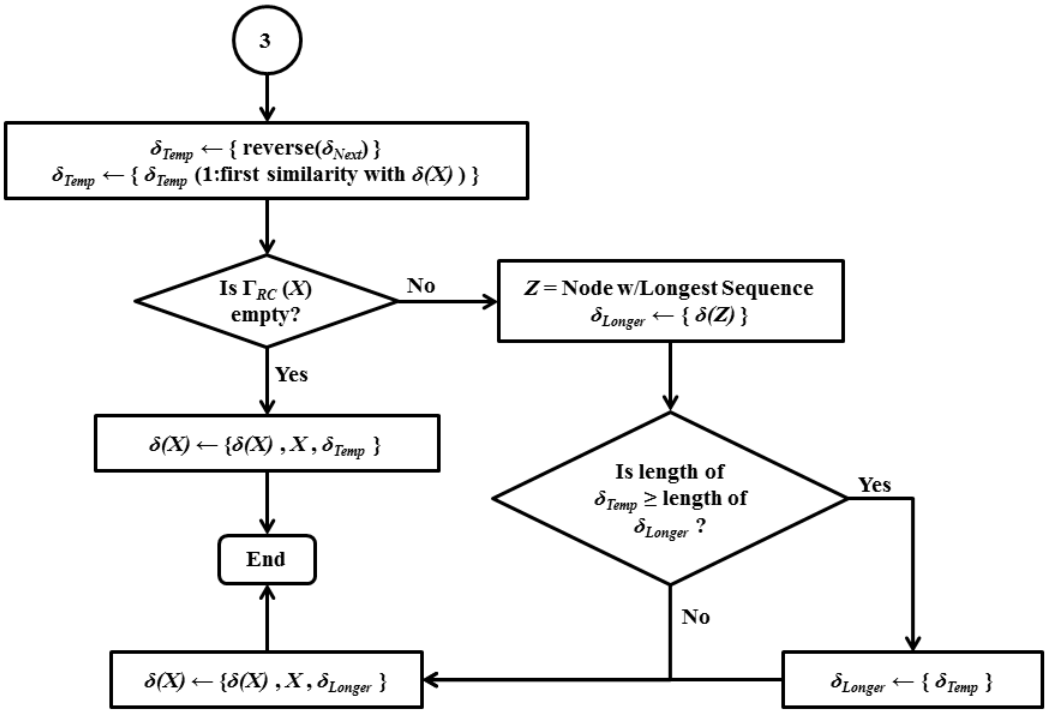

Figure 45. Third flow chart adjusting $\delta(X)$ for similarities.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SIMULATION SOURCE CODE

Source codes for the simulations conducted in Chapter IV of this thesis are provided in this appendix.

A. MATLAB VALIDATION OF R4SYN ESTIMATORS FILE

The MATLAB source code presented in this section can be used to test the R4Syn estimators presented in Chapter II and Chapter IV of this thesis. The code can only be run to simulate four nodes broadcasting for ten cycles for the one-hop scenario. For the multi-hop scenario, the code can only be run to simulate up to ten hops.

```
% Validation of R4Syn estimators

clc, clear all, close all

%initialize values
N = 4;  %number of nodes
J = 51; %total number of cycles needed for Node 2's calculations
S = 1000; %number of measurements
sigma = 1e-6;
%B = 10;   %number of hops            %un-comment for multi-hop
%TotalMSEskew = zeros(B,10);                %un-comment for multi-hop
SumMSEskew = zeros(1,10); SumMSEoffset = zeros(1,10);
SumCRLBskew = zeros(1,10); SumCRLBoffset = zeros(1,10);
%for rrr=1:S                                %un-comment for multi-hop
 % MH_ActualSkew = zeros(1,B);              %un-comment for multi-hop
 % MH_Actual_Skew = [];                     %un-comment for multi-hop
 % XXX = cell(B,1);                         %un-comment for multi-hop
 % for MH=1:B                               %un-comment for multi-hop
for iii=1:S                         %remove for multi-hop
  FinalMSEskew = []; %remove for multi-hop
  FinalMSEoffset = [];%remove for multi-hop
  FinalCRLBskew = [];         %remove for multi-hop
  FinalCRLBoffset = [];         %remove for multi-hop
  %set distance between nodes
  d = zeros(N,N);
  for i=1:N
    for k=1:N
      if i~=k && k>i
        d(i,k) = 90 + sqrt(5).*randn(1,1);
```

```
      d(k,i) = d(i,k);
    else
    end
  end
end
%processing times
xx = 100; yy = 500;
t_p = xx + (yy-xx).*rand(1,N);
t_proc = t_p*(1e-3);
%propagation times
c = 3e8;
t_prop = zeros(N,N);
for i = 1:N
  for k = 1:N
    t_prop(i,k) = d(i,k)./c;
    t_prop(k,i) = t_prop(i,k);
  end
end
%set random frequencies
r = -.25; q = .25;
F = r + (q-r).*rand(1,N);
f = 1 + F;
%relative skew matrix (saved for MSE calc)
RelSkew = ones(N,N);
for i = 1:N
  for k = 1:N
    RelSkew(i,k) = f(k)./f(i);
  end
end
sk = RelSkew;
Node1Clock = 0;
x = -10; y = 10;
RandOffset = x + (y-x).*rand(1,N-1);
StartOffset = Node1Clock + RandOffset;
%simulate first cycle of timestamps
TS = zeros(N,N);
for i = 1:N-1
  TS(i+1,1) = StartOffset(i) + t_prop(1,i+1)*sk(1,i+1);
end
TS(1,2) = Node1Clock + (t_proc(2) + 2*t_prop(1,2))*sk(1,2);
for i = 1:N-2
  TS(1,i+2) = TS(1,i+1) + (t_proc(i+2) + t_prop(1,i+2))*sk(1,i+2);
end
for i=2:N
  for k=2:N
```

```
    if i==k
      TS(i,k) = 0;
    elseif i == k-1
      TS(i,k) = TS(i,k-2) + (t_proc(i) + t_proc(k) + (2*t_prop(i,k)))*sk(i,k);
    else
      TS(i,k) = TS(i,k-1) + (t_proc(k) + t_prop(i,k))*sk(i,k);
    end
  end
end
%simulate the rest of the timestamps for J cycles
for h = 2 : J
  TS = [TS zeros(N,N)];
  for i=1:N
    for k = ((h-1)*N)+1 : h*N
      if i == k-(h-1)*N
        TS(i,k) = 0;
      elseif i == k -(N*(h-1)+1)
        TS(i,k) = TS(i,k-2) + (t_proc(i)+t_proc(i+1) + (2*t_prop(i,i+1)))*sk(i,i+1);
      elseif i == (k-1) - (h-2)*N
        TS(i,k) = TS(i,k-2) + (t_proc(1)+t_proc(i) + (2*t_prop(i,1)))*sk(i,1);
      else
        TS(i,k) = TS(i,k-1) + (t_proc(k-(h-1)*N) + t_prop(i, k-(h-1)*N))*sk(i,k-(h-1)*N);
      end
    end
  end
end
%initialize zero matrix for each node
for i = 1:N
  Node{i} = zeros(N,N);
  for k = 1:J-1
    Node{i} = [Node{i} zeros(N,N)];
  end
end
for i = 2:N
  Node{i}(i,1) = TS(i,1);
end
for k = 2:N
  for i=1:N
    if i~=k
      Node{i}(k,1:k-1) = TS(k,1:k-1);
      Node{i}(i,k) = TS(i,k);
    else continue
    end
  end
end
```

```
for h = 2 : J
 for k = 1:N
  for i=1:N
   if i~=k
    Node{i}(k,((h-1)*N+k)-3 : ((h-1)*N+k)-1) = TS(k,((h-1)*N+k)-3 : ((h-1)*N+k)-1);
    Node{i}(i,(h-1)*N+k) = TS(i,(h-1)*N+k);
   else continue
   end
  end
 end
end
%gathering values for offset/skew calculations
for gg=6:5:51
 for i=1:N
  values{i} = cell(N,2);
 end
 for i=1:N
  for k = 1 : gg * N
   if Node{i}(i,k) ~= 0
    for m=1:N
     if Node{i}(m,k)~=0 && m~=i
      values{i}{m,1} = [values{i}{m,1} Node{i}(m,k)];
      values{i}{m,2} = [values{i}{m,2} Node{i}(i,k)];
     else continue
     end
    end
   else continue
   end
  end
 end
 EstimatorSkew = ones(N,N);
 EstimatorOffset = zeros(N,N);
 for ii=1:N
  for kk=1:N
   u = values{ii}{kk,1};
   v = values{ii}{kk,2};
   if length(u)>1
    K = length(u);
    %calculate estimators per R4Syn
    skew = (sum(u)*sum(v) - K*(u*v')) / ((sum(v)).^2 - K*sum(v.^2));
    offset = (1/K) * ( sum(u)-(skew*sum(v)) );
    EstimatorSkew(ii,kk) = skew;
    EstimatorOffset(ii,kk) = offset;

    skewCRLB = (K*sigma^2) / ((K*sum(v.^2)) - ((sum(v)).^2)) ;
```

```
        offsetCRLB = ((sigma^2)*(sum(v.^2))) / ((K*sum(v.^2)) - ((sum(v)).^2)) ;
        CRLBskewMatrix(ii,kk) = skewCRLB;
        CRLBoffsetMatrix(ii,kk) = offsetCRLB;
       else continue
       end
     end
   end
% ######UN-COMMENT THE FOLLOWING SECTION FOR MULTI-HOP#######
   %XXX{MH} = [XXX{MH} EstimatorSkew(2,1)];
   %end
 % end
 % MSEskewHop=cell(B,1);
 %MH_Est_Skew = [];
 %for aa = 1:B
  %if aa == 1
  % MH_Est_Skew = XXX{1};
  %else
  %  MH_Est_Skew = MH_Est_Skew.*XXX{aa};
  % end
 %  MSEskewHop{aa} = (MH_ActualSkew(aa) - MH_Est_Skew).^2;
 %end
 %Z= zeros(B,10);
 %for i=1:B
  % Z(i,:) = MSEskewHop{i};
 %end
  % TotalMSEskew = TotalMSEskew + Z;
%end
%AverageMSEfreq = TotalMSEskew./S;
%figure()
%surf(AverageMSEfreq)
%set(gca, 'XScale', 'linear', 'YScale', 'linear', 'ZScale', 'log')
%xlabel('Number of Messages'), ylabel('Number of Hops'), zlabel('MSE')
%grid on
% #########END UN-COMMENTED SECTION FOR MULTI-HOP#########
   RelOffset = zeros(N,N);
   for i=1:N
     for p = (gg-1) * N: gg*N - 1
       for k=1:N
        if i==k
          RelOffset(i,k)=0;
        elseif TS(i,p)~=0 && TS(k,p)~=0
          RelOffset(i,k) = TS(k,p) - RelSkew(i,k)*TS(i,p);
        else continue
        end
       end
```

```
    end
  end
  MSEskew = (RelSkew - EstimatorSkew).^2;
  MSEoffset = (RelOffset - EstimatorOffset).^2;
  FinalMSEskew = [FinalMSEskew MSEskew(2,1)];
  FinalMSEoffset = [FinalMSEoffset MSEoffset(2,1)];
  FinalCRLBskew = [FinalCRLBskew CRLBskewMatrix(2,1)];
  FinalCRLBoffset = [FinalCRLBoffset CRLBoffsetMatrix(2,1)];
 end
 SumMSEskew = SumMSEskew + FinalMSEskew;
 SumMSEoffset = SumMSEoffset + FinalMSEoffset;
 SumCRLBskew = SumCRLBskew + FinalCRLBskew;
 SumCRLBoffset = SumCRLBoffset + FinalCRLBoffset;
end
AverageMSEskew = SumMSEskew./S;
AverageMSEoffset = SumMSEoffset./S;
AverageCRLBfreq = SumCRLBskew./S;
AverageCRLBoffset = SumCRLBoffset./S;
X = 10:10:100;
figure()
semilogy(X, AverageMSEskew, 'bo-', X, AverageCRLBfreq,'rx:')
xlabel('Number of Messages'), ylabel('MSE')
legend('MSE', 'CRLB', 'location', 'northeast')
figure()
semilogy(X, AverageMSEoffset,'bo-', X, AverageCRLBoffset,'rx:')
xlabel('Number of Messages'), ylabel('MSE (sec)')
legend('MSE', 'CRLB', 'location', 'northeast')
```

B. MATLAB BROADCAST SEQUENCING AND HYBRID TIME SYNCHRONIZATION SIMULATION FILE

The MATLAB source code presented in this section can be used to test the broadcast sequencing and hybrid time synchronization scheme presented in Chapter III and Chapter IV of this thesis. The broadcast sequencing code is run by changing the number of nodes, type of topology, and the number of topologies to the desired scenario. The hybrid time synchronization code is run by changing the topology type, the number of nodes, and the number of topologies to the desired scenario.

```
% Broadcast sequencing and hybrid time synchronization file
% un-comment and/or remove lines of code as instructed for the desired scenario
clc, clear all, close all

numTopol = 100;
C = 10;
```

```
% ######### delete the following for only broadcast sequencing #############
t_avg_prec = zeros(1,C); t_avg_1hop = zeros(1,C); t_avg_2hop = zeros(1,C); t_avg_3hop
= zeros(1,C); t_avg_4hop = zeros(1,C); t_avg_5hop = zeros(1,C); t_avg_6hop =
zeros(1,C); t_avg_7hop = zeros(1,C); t_avg_8hop = zeros(1,C); t_avg_9hop = zeros(1,C);
t_avg_10hop = zeros(1,C); t_avg_RSP = zeros(1,C);
ctprec=0; ct1hop=0; ct2hop=0; ct3hop=0; ct4hop=0; ct5hop=0; ct6hop=0; ct7hop=0;
ct8hop=0; ct9hop=0; ct10hop=0; ctRSP=0;
 % ######################################################################
%BP = 0;        un-comment for broadcast sequencing
%AP = 0;        un-comment for broadcast sequencing

for nT = 1:numTopol
  clear S alltimes CL column nodescansync nodetimes newtime TS seqIDs sync_max row
onecycle values
  close (figure(1))
  %% Input values
  N = 50;       % number of nodes in WSN
  maxrange = 100;   % maximum effective range of nodes (meters)
  topolsize = 'Sparse';
  idlength = 2;
  range = sqrt((maxrange.^2)/2);
  S(1).xd = 0; S(1).yd = 0;  % first node centered at origin
  hold on
  figure(1)
  plot(S(1).xd, S(1).yd, 'g*');
  nodeID = assignID(idlength);
  S(1).ID = {nodeID};
  uistack(text(S(1).xd, S(1).yd, S(1).ID,'EdgeColor','k', 'BackgroundColor', 'w'),'top');
  nodeIDlist = S(1).ID;
  %% Generate the WSN field and show topology
  if (strcmp(topolsize,'Sparse'))
    signx(1) = sign(randn(1));
    signy(1) = sign(randn(1));
    for i=2:N
      S(i).xd = S(i-1).xd + signx(i-1)*((range-range/3) + (range/3).*rand(1,1));
      S(i).yd = S(i-1).yd + signy(i-1)*((range-range/3) + (range/3).*rand(1,1));
      figure(1)
      plot(S(i).xd, S(i).yd, 'ro');
      nodeID = assignID(idlength);
      S(i).ID = {nodeID};
      while(sum(strcmp(S(i).ID, nodeIDlist))==1)
        nodeID = assignID(idlength);
        S(i).ID = {nodeID};
      end
      uistack(text(S(i).xd, S(i).yd, S(i).ID,'EdgeColor','k', 'BackgroundColor', 'w'),'top');
```

```
    nodeIDlist = [nodeIDlist S(i).ID];
    signx(i) = sign(randn(1));
    signy(i) = sign(randn(1));
    while signx(i) == -signx(i-1) && signy(i)== -signy(i-1)
      signx(i) = sign(randn(1));
      signy(i) = sign(randn(1));
    end
  end
elseif (strcmp(topolsize,'Dense'))
  for i=2:N
    S(i).xd = -range + (2*range).*rand(1,1);
    S(i).yd = -range + (2*range).*rand(1,1);
    figure(1)
    plot(S(i).xd, S(i).yd, 'r*');
    nodeID = assignID(idlength);
    S(i).ID = {nodeID};
    while(sum(strcmp(S(i).ID, nodeIDlist))==1)
      nodeID = assignID(idlength);
      S(i).ID = {nodeID};
    end
    uistack(text(S(i).xd, S(i).yd, S(i).ID,'EdgeColor','k', 'BackgroundColor', 'w'),'top');
    nodeIDlist = [nodeIDlist S(i).ID];
  end
end
xlabel('x-axis (meters)', 'FontSize', 12);    ylabel('y-axis (meters)', 'FontSize', 12);
for i=1:N
  S(i).numNeighbors = 0;
  for k=1:N
    if i~=k
      dist = sqrt(double((S(i).xd - S(k).xd)^2 + (S(i).yd - S(k).yd)^2));
      if dist < maxrange
        S(i).numNeighbors = S(i).numNeighbors + 1;
        figure(1)
        uistack(plot( [S(i).xd, S(k).xd], [S(i).yd, S(k).yd] ),'bottom');
        S(i).neighbors(S(i).numNeighbors) = S(k).ID;
      else continue
      end
    else continue
    end
  end
end
%% Initialize values for all nodes
for i=1:N
  S(i).receivedBeginPacket = 0; S(i).hopCount = 0; S(i).friendIDs = {};
S(i).sentToNeighbors = 0; S(i).parentIDs={}; S(i).childrenIDs={}; S(i).sequence={};
```

```
  end
  %% assign random node as starter node; send begin packet (flood network)
  starterNode = randi(N);
  uistack(text(S(starterNode).xd, S(starterNode).yd, S(starterNode).ID,'EdgeColor','k',
'BackgroundColor', 'y'),'top');
  S(starterNode).receivedBeginPacket=1;
  S(starterNode).sentToNeighbors=1;
  S(starterNode).parentIDs = S(starterNode).ID;
  S(starterNode).childrenIDs = S(starterNode).neighbors;
  for i=1:S(starterNode).numNeighbors
    for k=1:N
      if( char(S(k).ID) == char(S(starterNode).neighbors(i)) )
        S(k).receivedBeginPacket = 1;
        S(k).hopCount = S(k).hopCount + 1;
        S(k).parentsHopCount = 0;
        S(k).parentIDs = S(starterNode).ID;
        S(k).childrenIDs = S(k).neighbors;
        for jj=1:S(k).numNeighbors
          if(char(S(k).parentIDs) == char(S(k).neighbors(jj)))
            S(k).childrenIDs(jj)=[];
          else continue
          end
        end
      else continue
      end
    end
  end
  hold off
  refcount = 1;
  nodesNotified = 0;
  while (nodesNotified ~= N)
    for i = 1:N
      if (S(i).receivedBeginPacket == 1 && S(i).sentToNeighbors == 0 && S(i).hopCount
== refcount)
        S(i).sentToNeighbors = 1;
        for k=1:numel(S(i).childrenIDs)
          for m=1:N
            if( char(S(m).ID) == char(S(i).childrenIDs(k)))
              if(S(m).receivedBeginPacket==0)
                S(m).receivedBeginPacket = 1;
                if (isempty(S(m).parentIDs))
                  S(m).parentIDs = S(i).ID;
                  S(m).hopCount = S(i).hopCount + 1;
                  S(m).parentsHopCount = S(i).hopCount;
                  S(m).childrenIDs = S(m).neighbors;
```

```
            for jj=1:S(m).numNeighbors
              if(char(S(m).parentIDs) == char(S(m).neighbors(jj)))
                S(m).childrenIDs(jj)=[];
              else continue
              end
            end
          elseif (S(i).hopCount < S(m).parentsHopCount)
            S(m).parentIDs = S(i).ID;
            S(m).hopCount = S(i).hopCount + 1;
            S(m).parentsHopCount = S(i).hopCount;
            S(m).childrenIDs = S(m).neighbors;
            for jj=1:S(m).numNeighbors
              if(char(S(m).parentIDs) == char(S(m).neighbors(jj)))
                S(m).childrenIDs(jj)=[];
              else continue
              end
            end
          else continue
          end
        elseif (S(i).hopCount == S(m).parentsHopCount)
          S(m).parentIDs = [S(m).parentIDs S(i).ID];
          S(m).childrenIDs = S(m).neighbors;
          while(~isempty(intersect(S(m).childrenIDs, S(m).parentIDs)))
            for jj=1:numel(S(m).childrenIDs)
              for kk=1:numel(S(m).parentIDs)
                if (char(S(m).childrenIDs(jj)) == char(S(m).parentIDs(kk)))
                  break
                else continue
                end
              end
              if(char(S(m).childrenIDs(jj)) == char(S(m).parentIDs(kk)))
                S(m).childrenIDs(jj)=[];
                break
              end
            end
          end
        else continue
        end
      else continue
      end
    end
  end
 else continue
 end
end
```

```
    nodesNotified=0;
    for iii=1:N
      nodesNotified = nodesNotified + S(iii).receivedBeginPacket;
    end
    if (nodesNotified ~= N)
      refcount = refcount + 1;
    else
      break
    end
  end
  for i=1:N
    for k=1:numel(S(i).childrenIDs)
      for m=1:N
        if ( char(S(i).childrenIDs(k))== char(S(m).ID) )
          if (S(m).hopCount == S(i).hopCount) %if neighbor's hop count is same, they are
peers
            S(i).friendIDs = [S(i).friendIDs S(m).ID];
          else continue
          end
        else continue
        end
      end
    end
  end
  for i=1:N
    if( ~ isempty( S(i).friendIDs ) )
      while( ~ isempty( intersect( S(i).childrenIDs, S(i).friendIDs ) ) )
        for jj = 1 : numel( S(i).childrenIDs )
          for kk = 1 : numel( S(i).friendIDs )
            if( char( S(i).childrenIDs(jj) ) == char( S(i).friendIDs(kk) ) )
              break
            else continue
            end
          end
          if( char(S(i).childrenIDs(jj) ) == char( S(i).friendIDs(kk)) )
            S(i).childrenIDs(jj) = [];
            break
          end
        end
      end
    else continue
    end
  end
  for i=1:N
    S(i).numMasters = numel(S(i).parentIDs);
```

```
  S(i).numPeers = numel(S(i).friendIDs);
end
%% all nodes notified; begin sequence determination
for i=1:N
  rc(i) = S(i).hopCount;
end
  refcountB = max(rc);
for i=1:N
  S(i).recSeqPkt = 0;
end
nodesReceivedSeq = 0;
while (nodesReceivedSeq ~= N)
  for i = 1:N
    if( S(i).recSeqPkt == 0 && S(i).hopCount == refcountB )
      S(i).recSeqPkt = 1;
      if( isempty(S(i).childrenIDs) ) %if node has NO children
        S(i).sequence = S(i).ID;
      elseif ( numel(S(i).childrenIDs) == 1) %if node has ONE child
        for ii=1:N
          if( strcmp(S(ii).ID, S(i).childrenIDs))
            if( strcmp(S(ii).sequence(end), S(ii).ID))
              S(i).sequence = [S(ii).sequence S(i).ID];
            else
              for ss = 1:numel(S(ii).sequence)
                if( strcmp(S(ii).sequence(ss), S(ii).ID))
                  if(numel(S(ii).sequence(1:ss))>=numel(S(ii).sequence(ss:end)))
                    final = S(ii).sequence(1:ss);
                  else
                    final = fliplr(S(ii).sequence(ss:end));
                  end
                else continue
                end
              end
              S(i).sequence = [final S(i).ID];
            end
          else continue
          end
        end
      elseif ( numel(S(i).childrenIDs) > 1 ) %if node has more than one child
        childnumID = [];
        childrenseqs = [];
        for a=1:numel(S(i).childrenIDs)
          for h=1:N
            if( strcmp(S(i).childrenIDs(a), S(h).ID) )
              childnumID = [childnumID h];
```

```
            childrenseqs{a} = S(h).sequence;
          else continue
          end
        end
      end
      highestID = findlongestsequence(S,childnumID, childrenseqs);
      S(i).sequence = S(childnumID(highestID)).sequence;
      longerseq={};
      for ls=1:numel(S(childnumID(highestID)).sequence)
        if strcmp(S(childnumID(highestID)).sequence(ls), S(childnumID(highestID)).ID)
==0
          longerseq(ls) = S(childnumID(highestID)).sequence(ls);
        else break
        end
      end
      longerseq = [longerseq S(childnumID(highestID)).ID];
      childnumID(highestID) = [];
      remainingchildren={};
      for rk=1:length(childnumID)
        remainingchildren(rk) = S(childnumID(rk)).ID;
      end
      while numel(remainingchildren) ~= 0
        [lastnodeinseq] = S(i).sequence(end);
        for lg=1:N
          if( strcmp(S(lg).ID, lastnodeinseq) )
            lastnodeID = lg;
          else continue
          end
        end
        parentfriend = [S(lastnodeID).parentIDs S(lastnodeID).friendIDs];
        [remainingchildren,PF,S] = addparentfriend(S, N, remainingchildren,
parentfriend, i);
        [remainingchildren] = UpdateRemainingChildren(S, i, remainingchildren);
        if numel(remainingchildren)== 0
          S(i).sequence = [S(i).sequence S(i).ID];
          break
        else
        end
        nextchildID = [];
        nextchildseq = [];
        for aaa=1:numel(remainingchildren)
          if( isempty(intersect( remainingchildren(aaa), S(i).sequence )) )
            for ww=1:N
              if( strcmp(remainingchildren(aaa), S(ww).ID) )
                nextchildID = [nextchildID ww];
```

```
            nextchildseq{aaa} = S(ww).sequence;
          else continue
          end
        end
      else continue
      end
    end
     if(length(nextchildID) > 1)
      nexthighestID = findlongestsequence(S, nextchildID, nextchildseq);
    else
      nexthighestID = 1;
    end
    shorterseq={};
    for dd=1:numel(S(nextchildID(nexthighestID)).sequence)
      if( strcmp(S(nextchildID(nexthighestID)).sequence(dd),
S(nextchildID(nexthighestID)).ID) == 0)
        shorterseq(dd) = S(nextchildID(nexthighestID)).sequence(dd);
      else break
      end
    end
    shorterseq = [shorterseq S(nextchildID(nexthighestID)).ID];
    for tt=1:numel(remainingchildren)
      if( strcmp(S(nextchildID(nexthighestID)).ID, remainingchildren(tt)) )
        remainingchildren(tt)=[];
        break
      else
      end
    end
    [remainingchildren] = UpdateRemainingChildren(S, i, remainingchildren);
    if numel(remainingchildren)== 0
      PFnext = 0;
    else
      [lastnodeinseq] = S(nextchildID(nexthighestID)).sequence(end);
      for lg=1:N
        if( strcmp(S(lg).ID, lastnodeinseq) )
          lastnodeID = lg;
        else continue
        end
      end
      parentfriend = [S(lastnodeID).parentIDs S(lastnodeID).friendIDs];
      [remainingchildren, PFnext, S] = addparentfriend(S, N, remainingchildren,
parentfriend, nextchildID(nexthighestID));
    end
    if PF == 0 && PFnext == 0
      S(i).sequence = longerseq;
```

```
            S(nextchildID(nexthighestID)).sequence = shorterseq;
          elseif PF == 0 && PFnext == 1
            S(i).sequence = longerseq;
          elseif PF == 1 && PFnext == 0
            S(nextchildID(nexthighestID)).sequence = shorterseq;
          else
          end
          %adjusting for similarities
          %if there are NO similarities:
          if(isempty(intersect(S(i).sequence, S(nextchildID(nexthighestID)).sequence)))
            S(i).sequence = [S(i).sequence S(i).ID
fliplr(S(nextchildID(nexthighestID)).sequence)];
            [remainingchildren] = UpdateRemainingChildren(S, i, remainingchildren);
            if numel(remainingchildren)== 0
              break
            else
              [lastnodeinseq] = S(i).sequence(end);
              for lg=1:N
                if( strcmp(S(lg).ID, lastnodeinseq) )
                  lastnodeID = lg;
                else continue
                end
              end
              parentfriend = [S(lastnodeID).parentIDs S(lastnodeID).friendIDs];
              [remainingchildren,PF, S] = addparentfriend(S, N, remainingchildren,
parentfriend,i);
            end
            break
          else %there are similarities
            if(strcmp(S(i).sequence(1), S(nextchildID(nexthighestID)).sequence(1)))
              if (~isempty(intersect(S(i).sequence(2:end),
S(nextchildID(nexthighestID)).sequence(2:end))))
                nss = fliplr(S(i).sequence);
                temp = {};
                for yu = 1:numel(nss)
                  if isempty(intersect(nss(yu),S(nextchildID(nexthighestID)).sequence))
                    temp(yu) = nss(yu);
                  else break
                  end
                end
                S(i).sequence = [S(nextchildID(nexthighestID)).sequence, S(i).ID, temp];
                break
              else
                S(i).sequence(1) = [];
```

```
            S(i).sequence = [fliplr(S(i).sequence)
S(nextchildID(nexthighestID)).sequence];
            [remainingchildren] = UpdateRemainingChildren(S, i, remainingchildren);
            if numel(remainingchildren)== 0
              S(i).sequence = [S(i).sequence S(i).ID];
              break
            else
            end
          end
        elseif strcmp(S(i).sequence(end), S(nextchildID(nexthighestID)).sequence(1))
          if ~isempty(intersect(S(i).sequence(1:end-1),
S(nextchildID(nexthighestID)).sequence(2:end)))
            nss = fliplr(S(nextchildID(nexthighestID)).sequence);
            temp = {};
            for yu = 1:numel(nss)
              if isempty(intersect(nss(yu),S(i).sequence))
                temp(yu) = nss(yu);
              else break
              end
            end
            S(i).sequence = [S(i).sequence, S(i).ID, temp];
            break
          else
            S(i).sequence(end) = [];
            S(i).sequence = [S(i).sequence S(nextchildID(nexthighestID)).sequence];
            [remainingchildren] = UpdateRemainingChildren(S, i, remainingchildren)
            if numel(remainingchildren)== 0
              S(i).sequence = [S(i).sequence S(i).ID];
              break
            else
            end
          end
        else
          nss = fliplr(S(nextchildID(nexthighestID)).sequence);
          temp = {};
          for yu = 1:numel(nss)
            if isempty(intersect(nss(yu),S(i).sequence))
              temp(yu) = nss(yu);
            else break
            end
          end
          [remainingchildren] = UpdateRemainingChildren(S, i, remainingchildren);
          if numel(remainingchildren)==0
            S(i).sequence = [S(i).sequence, S(i).ID, temp];
            break
```

```
            else
              nextchildID = [];
              nextchildseq = [];
              for aaa=1:numel(remainingchildren)
                if( isempty(intersect( remainingchildren(aaa), S(i).sequence )) )
                  for ww=1:N
                    if( strcmp(remainingchildren(aaa), S(ww).ID) )
                      nextchildID = [nextchildID ww];
                      nextchildseq{aaa} = S(ww).sequence;
                    else continue
                    end
                  end
                else continue
                end
              end
              if length(nextchildID) > 1
                nexthighestID = findlongestsequence(S, nextchildID, nextchildseq);
              else
                nexthighestID = 1;
              end
              nextshorterseq={};
              for dd=1:numel(S(nextchildID(nexthighestID)).sequence)
                if( strcmp(S(nextchildID(nexthighestID)).sequence(dd),
S(nextchildID(nexthighestID)).ID) == 0)
                  nextshorterseq(dd) = S(nextchildID(nexthighestID)).sequence(dd);
                else break
                end
              end
              nextshorterseq = [nextshorterseq S(nextchildID(nexthighestID)).ID];
              longer = fliplr(nextshorterseq);
              if numel(temp) > numel(nextshorterseq)
                longer = temp;
              else
              end
              S(i).sequence = [S(i).sequence, S(i).ID, longer];
              break
            end
          end
        end
      end
    else
    end
  else
    continue
  end
```

```
    end
    nodesReceivedSeq=0;
    for iii=1:N
      nodesReceivedSeq = nodesReceivedSeq + S(iii).recSeqPkt;
    end
    if (nodesReceivedSeq ~= N)
      refcountB = refcountB - 1;
    else break
    end
  end
  SSq = S(starterNode).sequence;
  %preSeqCorrNum = numel(SSq); beforePerc = (preSeqCorrNum / N) *100; BP = BP +
beforePerc;                %un-comment for broadcast sequencing

  %% sequence correction
  for sck=1:N
    nodeIDnum=[];
    for i=1:numel(SSq)
      for k=1:N
        if(strcmp(SSq(i),S(k).ID))
          nodeIDnum = [nodeIDnum k];
        else continue
        end
      end
    end
    for i=2:length(nodeIDnum)
      if(~isempty(intersect(S(nodeIDnum(i-1)).ID, S(nodeIDnum(i)).neighbors)))
        continue
      else
      end
    end
    if(nodeIDnum(1) == nodeIDnum(end) )
      nodeIDnum(end)=[];
    else
    end
    for i=2:length(nodeIDnum)
      for k=1:S(nodeIDnum(i-1)).numNeighbors
        if(isempty(intersect(S(nodeIDnum(i-1)).neighbors(k), SSq)) &&
~isempty(intersect(S(nodeIDnum(i-1)).neighbors(k), S(nodeIDnum(i)).neighbors)))
          for m=1:N
            if(strcmp(S(nodeIDnum(i-1)).ID, SSq(m)))
              a=SSq(1:m);
              b=SSq(m+1:end);
              SSq=[a S(nodeIDnum(i-1)).neighbors(k) b];
              nodeIDnum=[];
```

```
            for ii=1:numel(SSq)
              for kk=1:N
                if(strcmp(SSq(ii),S(kk).ID))
                  nodeIDnum = [nodeIDnum kk];
                else continue
                end
              end
            end
            brea
          else continue
          end
        end
        break
      else continue
      end
    end
  end
  for po=1:S(nodeIDnum(end)).numNeighbors
    if (isempty(intersect(S(nodeIDnum(end)).neighbors(po), SSq)))
      SSq = [SSq S(nodeIDnum(end)).neighbors(po)];
      break
    else continue
    end
  end
  SSq = fliplr(SSq);
 end
 %postSeqCorrNum = numel(SSq); afterPerc = (postSeqCorrNum / N) *100; AP = AP +
afterPerc;                          %un-comment for broadcast sequencing
 %end                                                %un-comment for broadcast sequencing
%end                                                 %un-comment for broadcast sequencing
%numTrials = numTopol * N; BP_final = BP/numTrials; %un-comment
%AP_final = AP/numTrials; %un-comment for broadcast sequencing

%% start synchronization
for i=1:N
 for k=1:S(i).numNeighbors
  for m=1:N
   if(strcmp(S(i).neighbors(k),S(m).ID))
    S(i).neighIDs(k) = m;
   else continue
   end
  end
 end
end
for i=1:length(SSq)
```

```
  for k=1:N
   if(strcmp(SSq(i),S(k).ID))
     seqIDs(i) = k;
   else continue
   end
  end
end
qr = seqIDs; qr(end)=[]; qr(1)=[]; seqIDs = [seqIDs fliplr(qr)]; onecycle =
length(seqIDs); seqIDs = repmat(seqIDs,1,C); nodetimes = zeros(N,1); TS = zeros(N,1);
%set different frequencies:
r = -.25; q = .25;          F = r + (q-r).*rand(1,N);          f = 1 + F;
%relative skew matrix (saved for MSE calc)
RelSkew = ones(N,N);
for i = 1:N
  for k = 1:N
    RelSkew(i,k) = f(k)./f(i);
  end
end
sk = RelSkew;
%set different offsets:
Node1Clock = 0;    x = -10; y = 10;    RandOffset = x + (y-x).*rand(1, N);
StartOffset = Node1Clock + RandOffset;
c = 3e8; xx = 100; yy = 500;  t_p = xx + (yy-xx).*rand(1,N);  t_proc = t_p*(1e-3);
%builds first column of timestamps:
for i=1:N
  if ~isempty(intersect(i, S(seqIDs(1)).neighIDs))
    dist = sqrt(double((S(seqIDs(1)).xd - S(i).xd)^2 + (S(seqIDs(1)).yd - S(i).yd)^2));
    t_prop = dist/c;    alltimes = t_prop;
    nodetimes(i,1) = sk(seqIDs(1),i)*alltimes + StartOffset(i);
    TS(i,1) = 1;
  else
    dist = sqrt(double((S(seqIDs(1)).xd - S(seqIDs(2)).xd)^2 + (S(seqIDs(1)).yd -
S(seqIDs(2)).yd)^2));
    t_prop = dist/c;        alltimes = t_prop;
    nodetimes(i,1) = sk(seqIDs(1),seqIDs(2))*alltimes + StartOffset(i);
    if seqIDs(1) == i
      TS(i,1) = 2;
    else
    end
  end
end
%builds remaining number of timestamps except last column
for k=2:length(seqIDs(2:end))
  for i=1:N
    if ~isempty(intersect(i, S(seqIDs(k)).neighIDs))
```

```
      dist = sqrt(double((S(seqIDs(k)).xd - S(i).xd)^2 + (S(seqIDs(k)).yd - S(i).yd)^2));
      t_prop = dist/c; alltimes = t_prop + t_proc(i);
      nodetimes(i,k) = sk(seqIDs(k),i)*alltimes + nodetimes(i,k-1);
      TS(i,k) = 1; %received beacons
    else
      dist = sqrt(double((S(seqIDs(k)).xd - S(seqIDs(k+1)).xd)^2 + (S(seqIDs(k)).yd -
S(seqIDs(k+1)).yd)^2));
      t_prop = dist/c; alltimes = t_prop + t_proc(i);
      nodetimes(i,k) = sk(seqIDs(k),seqIDs(k+1))*alltimes + nodetimes(i,k-1);
      if seqIDs(k) == i
        TS(i,k) = 2; %sender
      else
      end
    end
  end
end
%builds last column of timestamps:
lastk = length(seqIDs);   nodetimes = [nodetimes zeros(N,1)];        TS = [TS zeros(N,1)];
for i=1:N
  if ~isempty(intersect(i, S(seqIDs(end)).neighIDs))
    dist = sqrt(double((S(seqIDs(end)).xd - S(i).xd)^2 + (S(seqIDs(end)).yd - S(i).yd)^2));
    t_prop = dist/c;       alltimes = t_prop + t_proc(i);
    nodetimes(i,lastk) = sk(seqIDs(end),i)*alltimes + nodetimes(i,lastk-1);
    TS(i,lastk) = 1; %received beacons
  else
    dist = sqrt(double((S(seqIDs(end)).xd - S(seqIDs(1)).xd)^2 + (S(seqIDs(end)).yd -
S(seqIDs(1)).yd)^2));
    t_prop = dist/c;        alltimes = t_prop + t_proc(i);
    nodetimes(i,lastk) = sk(seqIDs(end),seqIDs(1))*alltimes + nodetimes(i,lastk-1);
    if seqIDs(end) == i
      TS(i,lastk) = 2; %sender
    else
    end
  end
end
avg_prec = []; avg_1hop = []; avg_2hop = []; avg_3hop = []; avg_4hop = []; avg_5hop =
[]; avg_6hop = []; avg_7hop = []; avg_8hop = []; avg_9hop = []; avg_10hop = [];
avg_RSP = [];
for CL = onecycle : onecycle : length(seqIDs)
  %finds master reference node:
  synctotal=[];
  for row=1:N
    nodescansync = 0;
    for column = 1 : CL
      if(TS(row, column)==1)
```

```
      for lin=1:N
       if TS(lin,column)==1 && lin==row
         continue
       elseif TS(lin, column)==1
         nodescansync = nodescansync + 1;
       else continue
       end
      end
    else continue
    end
  end
  synctotal(row) = nodescansync;
end
sync_max = max(synctotal);
MastRefNode = find(sync_max==synctotal);
if length(MastRefNode)>1
  MastRefNode = MastRefNode(1);
end
for i=1:N
  values{i} = cell(N,2);
end
%gather samples from same beacons:
for i=1:N
  for k = 1 : CL
    if TS(i,k) == 1
      for m=1:N
       if TS(m,k)==1 && m~=i
         values{i}{m,1} = [values{i}{m,1} nodetimes(m,k)];
         values{i}{m,2} = [values{i}{m,2} nodetimes(i,k)];
       else continue
       end
      end
    else continue
    end
  end
end
EstSkew = ones(N,N); EstOffset = zeros(N,N);
for ii=1:N
  for kk=1:N
    u = values{ii}{kk,1}; v = values{ii}{kk,2};
    if length(u) > 1
      K = length(u);
      skew = (sum(u)*sum(v) - K*(u*v')) / ((sum(v)).^2 - K*sum(v.^2));
      offset = (1/K) * ( sum(u)-(skew*sum(v)) );
      EstSkew(ii,kk) = skew; EstOffset(ii,kk) = offset;
```

```
      else continue
      end
    end
  end
  for i=1:N
    S(i).synced = 0;
  end
  S(MastRefNode).synced = 1;
  S(MastRefNode).newclock = nodetimes(MastRefNode,CL);
  prec = []; R4Syn_1hop = []; R4Syn_2hop = []; R4Syn_3hop = []; R4Syn_4hop = [];
R4Syn_5hop = []; R4Syn_6hop = []; R4Syn_7hop = []; R4Syn_8hop = []; R4Syn_9hop =
[]; R4Syn_10hop = []; RSP = [];
  %Execute R4Syn:
  for i=1:N
    if MastRefNode == i
      continue
    elseif EstOffset(i,MastRefNode) ~= 0
      %executes 1-hop R4Syn
      newtime = EstSkew(i,MastRefNode) * nodetimes(i,CL) + EstOffset(i,MastRefNode);
      S(i).newclock = newtime;  S(i).synced = 1;
      p = abs(newtime - nodetimes(MastRefNode,CL));
      S(i).precision = p; prec = [prec, p];
      R4Syn_1hop = [R4Syn_1hop, p];
    else
      %2-Hop:
      for k=1:N
        if EstOffset(i,k) ~= 0 && EstOffset(k,MastRefNode) ~= 0
          % executes 2-hop R4Syn
          totfreq = EstSkew(i,k) * EstSkew(k,MastRefNode);
          totoffset = EstSkew(k,MastRefNode) * EstOffset(i,k) +
EstOffset(k,MastRefNode);
          newtime = totfreq * nodetimes(i,CL) + totoffset;
          S(i).newclock = newtime; S(i).synced = 1;
          p = abs(newtime - nodetimes(MastRefNode,CL));
          S(i).precision = p; prec = [prec, p];
          R4Syn_2hop = [R4Syn_2hop, p];
          break
        else continue
        end
      end
      if S(i).synced==1
        continue
      end
      % Lines of code for 3-hop through 9-hop R4Syn are omitted to reduce repetitive lines
that can be extrapolated by the 10-hop code shown below:
```

```
%10-Hop:
for k=1:N
 if EstOffset(i,k) ~= 0
  for l=1:N
   if EstOffset(k,l) ~= 0
    for g=1:N
     if EstOffset(l,g) ~= 0
      for h=1:N
       if EstOffset(g,h) ~= 0
        for d=1:N
         if EstOffset(h,d) ~= 0
          for r=1:N
           if EstOffset(d,r) ~= 0
            for t=1:N
             if EstOffset(r,t) ~= 0
              for w=1:N
               if EstOffset(t,w)~=0
                for z=1:N
                 if EstOffset(w,z)~=0 && EstOffset(z,MastRefNode) ~= 0
                  % executes 10-hop R4Syn
                  totfreq =
EstSkew(i,k)*EstSkew(k,l)*EstSkew(l,g)*EstSkew(g,h)*EstSkew(h,d)*EstSkew(d,r)*Est
Skew(r,t)*EstSkew(t,w)*EstSkew(w,z)*EstSkew(z,MastRefNode);
                  totoffset =
EstOffset(i,k)*EstSkew(k,l)*EstSkew(l,g)*EstSkew(g,h)*EstSkew(h,d)*EstSkew(d,r)*Es
tSkew(r,t)*EstSkew(t,w)*EstSkew(w,z)*EstSkew(z,MastRefNode)...
                                    +
EstOffset(k,l)*EstSkew(l,g)*EstSkew(g,h)*EstSkew(h,d)*EstSkew(d,r)*EstSkew(r,t)*Es
tSkew(t,w)*EstSkew(w,z)*EstSkew(z,MastRefNode)...
                                      +
EstOffset(l,g)*EstSkew(g,h)*EstSkew(h,d)*EstSkew(d,r)*EstSkew(r,t)*EstSkew(t,w)*Es
tSkew(w,z)*EstSkew(z,MastRefNode)...
                                        +
EstOffset(g,h)*EstSkew(h,d)*EstSkew(d,r)*EstSkew(r,t)*EstSkew(t,w)*EstSkew(w,z)*E
stSkew(z,MastRefNode)...
                                          +
EstOffset(h,d)*EstSkew(d,r)*EstSkew(r,t)*EstSkew(t,w)*EstSkew(w,z)*EstSkew(z,Mast
RefNode)...
                                            +
EstOffset(d,r)*EstSkew(r,t)*EstSkew(t,w)*EstSkew(w,z)*EstSkew(z,MastRefNode)...
                                              +
EstOffset(r,t)*EstSkew(t,w)*EstSkew(w,z)*EstSkew(z,MastRefNode)...
                                                +
EstOffset(t,w)*EstSkew(w,z)*EstSkew(z,MastRefNode)...
```

```
                                                                +
EstOffset(w,z)*EstSkew(z,MastRefNode)...
                                                                    +
EstOffset(z,MastRefNode);
        newtime = totfreq * nodetimes(i,CL) + totoffset;
        S(i).newclock = newtime;
        S(i).synced = 1;
        p = abs(newtime - nodetimes(MastRefNode,CL));
        S(i).precision = p;
        prec = [prec, p];
        R4Syn_10hop = [R4Syn_10hop, p];
                                  break
                                else continue
                                end
                              end
                                          % ### repeat 8 more times###
                              if S(i).synced==1
                                break
                              end
                            else continue
                            end
                          end
                                          % ######################
  end
  %Execute RSP:
  for i=1:N
    if S(i).synced == 0
      for k=1:S(i).numNeighbors
        if S(S(i).neighIDs(k)).synced == 1
          dist = sqrt(double((S(i).xd - S(S(i).neighIDs(k)).xd)^2 + (S(i).yd -
S(S(i).neighIDs(k)).yd)^2));
        t_prop = dist/c;
        T1 = S(S(i).neighIDs(k)).newclock;
        T3 = T1 + t_proc(S(i).neighIDs(k));
        T2 = T1 + t_prop*sk(i,S(i).neighIDs(k));
        T4 = T3 + t_prop*sk(i,S(i).neighIDs(k));
        reftime = T3 + t_prop + t_proc(S(i).neighIDs(k));
        nodeti = T4 + t_proc(i);
        RSPskew = (T3-T1)/(T4-T2);
        RSPoffset = (T1*T4 - T2*T3) / (T4-T2);
        newtime = RSPskew*nodeti + RSPoffset;
        S(i).newclock = newtime;  S(i).synced = 1;
        p = abs(newtime - reftime);
        S(i).precision = p;
        prec = [prec, p]; RSP = [RSP, p];
```

```
        break
       else continue
       end
      end
    else continue
    end
  end
  AP = mean(prec); AR4S1 = mean(R4Syn_1hop); % repeat for hops 2 through 9
  AR4S10 = mean(R4Syn_10hop); ARSP = mean(RSP);
  avg_prec = [avg_prec, AP]; avg_1hop = [avg_1hop, AR4S1]; % repeat for hops 2
through 9
  avg_10hop = [avg_10hop, AR4S10]; avg_RSP = [avg_RSP, ARSP];
end
if isnan(avg_prec) == zeros(1,C)
  if avg_prec == zeros(1,C)
    avg_prec = zeros(1,C);
  else
    ctprec = ctprec + 1;
  end
else
  avg_prec = zeros(1,C);
end
% ###the following lines of code are repeated for up to 9 hops####
if isnan(avg_1hop) == zeros(1,C)
  if avg_1hop == zeros(1,C)
    avg_1hop = zeros(1,C);
  else
    ct1hop = ct1hop + 1;
  end
else
  avg_1hop = zeros(1,C);
end
% ####################################################
if isnan(avg_10hop) == zeros(1,C)
  if avg_10hop == zeros(1,C)
    avg_10hop = zeros(1,C);
  else
    ct10hop = ct10hop + 1;
  end
else
  avg_10hop = zeros(1,C);
end
if isnan(avg_RSP) == zeros(1,C)
  if avg_RSP == zeros(1,C)
    avg_RSP = zeros(1,C);
```

```
  else
    ctRSP = ctRSP + 1;
  end
else
  avg_RSP = zeros(1,C);
end
t_avg_prec = t_avg_prec + avg_prec;
t_avg_1hop = t_avg_1hop + avg_1hop; %repeat for hops 2 through 9
t_avg_10hop = t_avg_10hop + avg_10hop;
t_avg_RSP = t_avg_RSP + avg_RSP;
TotNumNeighbors = 0;
for i=1:N
  TotNumNeighbors = TotNumNeighbors + S(i).numNeighbors;
end
% Plot Average Number of Neighbors vs Precision
AvgNumNeigh = TotNumNeighbors / N;
NetAvgPrec= mean(avg_prec);
hold on
figure(2)
semilogy(AvgNumNeigh, NetAvgPrec, '*')
xlabel('Average Number of Neighbors per Node', 'FontSize',12,'FontWeight', 'bold')
ylabel('Precision (seconds)', 'FontSize',12,'FontWeight', 'bold')
xlim([0 N])
hold off
end
hold on
figure(3)
xaxis = 1:1:C;
if ct10hop ~=0
  semilogy(xaxis,t_avg_prec/ctprec,'b', xaxis, t_avg_RSP/ctRSP,'k',
xaxis,t_avg_1hop/ct1hop,'g', xaxis,t_avg_2hop/ct2hop,'r', xaxis,t_avg_3hop/ct3hop,'c',
xaxis, t_avg_4hop/ct4hop,'m', xaxis,t_avg_5hop/ct5hop,'b:',
xaxis,t_avg_6hop/ct6hop,'c:', xaxis,t_avg_7hop/ct7hop,'g:',
xaxis,t_avg_8hop/ct8hop,'r:', xaxis,t_avg_9hop/ct9hop,'k:',
xaxis,t_avg_10hop/ct10hop,'m:', 'LineWidth', 2)
  legend('Network Average', 'RSP Average', 'R4Syn 1-Hop Average', 'R4Syn 2-Hop
Average','R4Syn 3-Hop Average', 'R4Syn 4-Hop Average', 'R4Syn 5-Hop Average',
'R4Syn 6-Hop Average', 'R4Syn 7-Hop Average', 'R4Syn 8-Hop Average', 'R4Syn 9-
Hop Average', 'R4Syn 10-Hop Average', -1 )
%repeat pattern for hops 2 through 9 for plotting
elseif ct1hop ~=0
  semilogy(xaxis,t_avg_prec/ctprec,'b', xaxis, t_avg_RSP/ctRSP,'k',
xaxis,t_avg_1hop/ct1hop,'r*', 'LineWidth', 2)
  legend('Network Average', 'RSP Average', 'R4Syn 1-Hop Average' )
end
```

```
xlabel('Number of Cycles', 'FontSize',12,'FontWeight', 'bold')
ylabel('Precision (seconds)', 'FontSize',12,'FontWeight', 'bold')
hold off
% Plot Number of Hops vs Precision:
if ct10hop ~=0
  figure(4)
semilogy(1,mean(t_avg_1hop/ct1hop),'b*',2,mean(t_avg_2hop/ct2hop),'b*',3,mean(t_av
g_3hop/ct3hop),'b*',4,mean(t_avg_4hop/ct4hop),'b*',5,mean(t_avg_5hop/ct5hop),'b*',6
,mean(t_avg_6hop/ct6hop),'b*',7,mean(t_avg_7hop/ct7hop),'b*',8,mean(t_avg_8hop/ct8
hop),'b*',9,mean(t_avg_9hop/ct9hop),'b*',10,mean(t_avg_10hop/ct10hop),'b*','LineWi
dth', 2)
  xlabel('Number of Hops', 'FontSize',12,'FontWeight', 'bold')
  ylabel('Precision (seconds)', 'FontSize',12,'FontWeight', 'bold')
%repeat pattern for hops 2 through 9 for plotting
elseif ct1hop ~=0
  figure(4)
  semilogy(1,mean(t_avg_1hop/ct1hop),'b*','LineWidth', 2)
  xlabel('Number of Hops', 'FontSize',12,'FontWeight', 'bold')
  ylabel('Precision (seconds)', 'FontSize',12,'FontWeight', 'bold')
end
```

C. MATLAB ASSIGN ID FUNCTION

```
function [ nodeID ] = assignID(idlength)
  symbols = ['A':'Z' '0':'9'];
  nums = randi(numel(symbols),[1 idlength]);
  nodeID = symbols (nums);
end
```

D. MATLAB FIND LONGEST SEQUENCE FUNCTION

```
function [ highestID ] = findlongestsequence(S, childnumID, childrenseqs)
  highestID = 1;
  for b=2:length(childnumID) %finds child with longest sequence
    if( numel(childrenseqs{b}) > numel(childrenseqs{highestID}) )
      highestID = b;
    elseif( numel(childrenseqs{b}) == numel(childrenseqs{highestID}) )
      %comparing number of masters
      if ( S(childnumID(b)).numMasters < S(childnumID(highestID)).numMasters )
        highestID = b;
      elseif (S(childnumID(b)).numMasters==S(childnumID(highestID)).numMasters)
        %comparing number of peers
        if ( S(childnumID(b)).numPeers > S(childnumID(highestID)).numPeers )
          highestID = b;
        elseif ( S(childnumID(b)).numPeers == S(childnumID(highestID)).numPeers )
          %or random
```

```
        highestID = b;
      else continue
      end
    else continue
    end
  else continue
  end
 end
end
```

E. MATLAB ADD PARENT/FRIEND FUNCTION

```
function [remainingchildren, PF, S] = addparentfriend(S, N, remainingchildren,
parentfriend,i)
 PF=0;
 while ~isempty( intersect( remainingchildren, parentfriend ))
  shortnode = []; shortestpeerseq = [];
  for z=1:numel(remainingchildren)
   for u=1:numel(parentfriend)
    if (strcmp(remainingchildren(z), parentfriend(u)))
     for qq=1:N
      if(strcmp(remainingchildren(z), S(qq).ID))
       shortnode = [shortnode qq];
      else continue
      end
     end
    else continue
    end
   end
  end
  for zz=1:length(shortnode)
   shortestpeerseq{zz} = S(shortnode(zz)).sequence;
  end
  %finding shortest child sequence:
  shortestpeerID = 1;
  if length(shortnode) >= 2
   for xx=2:length(shortnode)
    if( numel(shortestpeerseq{xx}) < numel(shortestpeerseq{shortestpeerID}))
     shortestpeerID = xx;
    elseif numel(shortestpeerseq{xx})==numel(shortestpeerseq{shortestpeerID}))
     if ( S(shortnode(xx)).numPeers >= S(shortnode(shortestpeerID)).numPeers )
      shortestpeerID = xx;
     else
      shortestpeerID = shortestpeerID;
     end
    else
```

```
      shortestpeerID = shortestpeerID;
    end
  end
else
  shortestpeerID = 1;
end
if( isempty(intersect(S(shortnode(shortestpeerID)).ID, S(i).sequence)))
  S(i).sequence = [S(i).sequence, S(shortnode(shortestpeerID)).ID];
  PF = 1;
  for tt=1:numel(remainingchildren)
    if( strcmp(S(shortnode(shortestpeerID)).ID, remainingchildren(tt)) )
      remainingchildren(tt)=[];
      break
    else continue
    end
  end
  if numel(remainingchildren) == 0
    break
  else
    lastnodeinseq = S(i).sequence(end);
    for lg=1:N
      if( strcmp(S(lg).ID, lastnodeinseq) == 1)
        lastnodeID = lg;
      else continue
      end
    end
    parentfriend = [S(lastnodeID).parentIDs S(lastnodeID).friendIDs];
  end
else
  for tt=1:numel(remainingchildren)
    if( strcmp(S(shortnode(shortestpeerID)).ID, remainingchildren(tt)) )
      remainingchildren(tt)=[];
      break
    else continue
    end
  end
  if numel(remainingchildren) == 0
    break
  else
    lastnodeinseq = S(i).sequence(end);
    for lg=1:N
      if( strcmp(S(lg).ID, lastnodeinseq) == 1)
        lastnodeID = lg;
      else continue
      end
```

```
      end
      parentfriend = [S(lastnodeID).parentIDs S(lastnodeID).friendIDs];
    end
  end
 end
end
```

F. MATLAB UPDATE REMAINING CHILDREN FUNCTION

```
function [remainingchildren] = UpdateRemainingChildren(S, i, remainingchildren)
rks = {};
for k=1:numel(remainingchildren)
 if isempty(intersect(remainingchildren(k), S(i).sequence))
   rks = [rks remainingchildren(k)];
 else continue
 end
end
remainingchildren = rks;
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, Aug. 2002.

[2] W. Sun, Z. Yang, X. Zhang, and Y. Liu, "Energy-efficient neighbor discovery in mobile ad hoc and wireless sensor networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1448–1459, Mar. 2014.

[3] Y. Wu, Q. Chaudhair, and E. Serpedin, "Clock synchronization of wireless sensor networks," *IEEE Signal Processing Mag.*, vol. 28, no. 1, pp. 124–138, Jan. 2011.

[4] S. Ganeriwal, R. Kumar and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the First International Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, 2003, pp. 138–149.

[5] V. Claesso, H. Lonn, and N. Suri, "Efficient TDMA synchronization for distributed embedded systems," in *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, New Orleans, LA, 2001, pp. 198–201.

[6] D. Allan and M. Weiss, "Accurate time and frequency transfer during common-view of a GPS satellite," in *Proceedings of the 34thd Annual Frequency Control Symposium*, Fort Monmouth, NJ, 1980, pp. 334–346.

[7] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.

[8] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *Proceedings of the Second International Conference on Embedded Networked Sensor Systems*, Baltimore, MD, 2004, pp. 39–49.

[9] J. Sheu, W. Hu, and J. Lin, "Ratio-based time synchronization protocol in wireless sensor networks," *Telecommun. Syst.*, vol. 39, no. 1, pp. 25–35, Sep. 2008.

[10] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, 2002, pp. 147–163.

[11] D. Djenouri, "R4Syn: relative referenceless receiver/receiver time synchronization in wireless sensor networks," *IEEE Signal Process. Lett.*, vol. 19, no. 4, pp. 175–178, Apr. 2012.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California